國立交通大學

資訊科學與工程研究所

碩 士 論 文

超 橢 圓 曲 線 密 碼 攻 擊 之 研 究

A Study on Index Calculus Algorithms
for Hyperelliptic Curves

研 究 生：林家瑋

指導教授：陳榮傑　博士

中 華 民 國　九 十 六　年 六 月

超 橢 圓 曲 線 密 碼 攻 擊 之 研 究

A Study on Index Calculus Algorithms for Hyperelliptic Curves

研 究 生：林家瑋　　Student：Chia-Wei Lin

指導教授：陳榮傑 博士　　Advisor：Dr. Rong-Jaye Chen

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

June 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年六月

超　橢　圓　曲　線　密　碼　攻　擊　之　研　究

學生：林家瑋　　指導教授：陳榮傑博士

國立交通大學資訊科學與工程學研究所碩士班

摘　　要

1989 年 Koblitz 利用定義在有限域的超橢圓曲線上的 Jacobian 加法群，基於超橢圓曲線離散對數問題的困難度，提出了超橢圓曲線密碼系統。在含有 q 個元素的有限域 F_q 中，虧格(genus)為 g 的超橢圓曲線，其中形成離散對數問題的加法群大小為 $O(q^g)$，大於橢圓曲線加法群 $O(q)$。而且小虧格的超橢圓曲線亦無時間複雜度為次指數的攻擊法，因此適當的設定超橢圓曲線密碼系統將可使用比橢圓曲線密碼系統更短的密鑰，來達到相同的安全度。

目前 index calculus 攻擊法在虧格 g 足夠大時，呈現次指數的時間複雜度。當虧格不大時，一般的生日攻擊法為 $O(q^{\frac{g}{2}})$，而一般的 index calculus 為 $O(q^2)$。Thériault 的 index calculus 演算法加入"大質數"的概念，時間複雜度降為 $O(q^{2-\frac{4}{2g-1}})$；而 Gaudry 等人利用兩個"大質數"的 index calculus 攻擊法變形，則時間複雜度更進一步改進為 $O(q^{2-\frac{2}{g}})$。本文將針對小虧格的超橢圓曲線離散對數問題，實作並改進 index calculus 攻擊法。我們亦提出一個更快的演算法來解虧格為 2 的超橢圓曲線離散對數問題，其時間複雜度為 O(q)。

關鍵字: 超橢圓曲線密碼系統、超橢圓曲線離散對數問題、index calculus

A Study on Index Calculus Algorithms for Hyperelliptic Curves

Student：Chia-Wei Lin　　Advisors：Dr. Rong-Jaye Chen

Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University

ABSTRACT

In 1989, Koblitz proposed using the Jacobian of a hyperelliptic curve defined over a finite field to implement discrete logarithm cryptographic protocols. The discrete logarithm problem of the Jacobian is called hyperelliptic curve discrete logarithm problem (HCDLP). For a hyperelliptic curve of genus g over the finite field F_q, the group order of the Jacobian is $O(q^g)$ which is larger than that of the additive group ,which is $O(q)$, in an elliptic curve over F_q. Since there is no subexponential algorithm to solve HCDLP of small genus, hyperelliptic curve cryptosystem under applicable setting requires shorter key size than elliptic curve cryptosystem to achieve the same security level.

When genus g is large enough, the index calculus attack has subexponential time complexity. For small genus HCDLP, the algorithms based on birthday paradox is of time complexity $O(q^{\frac{g}{2}})$, and the basic index calculus attack is $O(q^2)$. Thériault improves it by using the large prime method, and get a running time of $O(q^{2-\frac{4}{2g-1}})$. Furthermore, Gaudry et al use a double large prime variation for small genus hyperelliptic index calculus, and the time complexity is $O(q^{2-\frac{2}{g}})$. In this thesis, we focus on the hyperelliptic curve discrete logarithm problem of small genus, implement and improve index calculus and its variations. We propose a faster algorithm for solving genus 2 HCDLP which time complexity is O(q).

Keywords: hyperelliptic curve cryptosystem, HCDLP, index calculus

致　　謝

首先誠摯的感謝指導教授陳榮傑教授，老師悉心的教導使我得以一窺超橢圓曲線密碼學的深奧，不時的討論並指點我正確的方向，使我在求學期間獲益匪淺，也讓這篇碩士論文能夠順利完成。也謝謝洪國寶教授、張仁俊教授、胡鈞祥博士擔任我的口試委員，在口試中給予的指正與建議，使得論文能更加完整。感謝師母李惠慈女士給予我英文寫作上的建議，使論文能更流暢通順。

感謝實驗室的志賢學長，及學弟妹們：輔國、用翔、佩娟，謝謝你們的陪伴使我的研究生活更加充實，還有準備口試期間的幫忙，使我能夠更充分準備。

最後我也要謝謝我的家人，感謝父母在求學期間多年來的栽培，謝謝弟弟和阿姨不時的關心，有你們的支持使我能夠無後顧之憂的專心求學、完成論文，謹以此文獻給我摯愛的家人。

Contents

List of Figures

List of Tables

Notation

The following notation is used throughout this thesis.

K	finite field
$\overline{K}$	algebraic closure of a finite field K
F_q	finite field of size $q = p^m$ for some prime p
q	size of the finite field F_q
g	genus of a hyperelliptic curve
$\mathbf{D^0}$	group of divisors on hyperelliptic curve of degree zero
P	group of principal divisors
J	quotient group $\mathbf{J=D^0/P}$
div(a,b)	a divisor denoted by Mumford representation with two polynomials a, b

Chapter 1

Introduction

1.1 History

Since the public-key cryptosystems have been invented in 1970s, there are several important public-key cryptosystems of which the security is based on the intractability of discrete logarithm problem (DLP) over a finite abelian group. Elliptic curve cryptography (ECC) [19] is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields. The use of elliptic curves in cryptography was suggested independently by Neal Koblitz and Victor S. Miller in 1985. There is no sub-exponential time algorithm to solve elliptic curve DLP (ECDLP), hence the main advantage of ECC is its smaller key size. A 160-bit key in ECC is considered to be as secure as 1024-key in RSA. As we can see in Table 1.1, ECC key size is much smaller than those of other public-key cryptosystems. Therefore ECC can be implemented efficiently and securely with smaller key size, and is ideally suitable for resource-constrained environments such as smart cards, cell phones, and PDAs.

However, hyperelliptic curve cryptosystems offer even smaller key size. In 1989, Koblitz [20] proposed using the Jacobian of a hyperelliptic curve defined over a finite field to implement discrete logarithm cryptographic protocols. Hyperelliptic curves are a special class of algebraic curves and can be viewed as generalizations of elliptic curves. There are hyperelliptic curves of every genus $g \geqq 1$. A hyperelliptic curve of genus $g = 1$ is an elliptic curve. There is no known

subexponential algorithm for hyperelliptic curves of small genus, and the Jacobian of a hyperelliptic curve of genus g defined over a finite field F_q has group order $O(q^g)$. Hence, the advantage of hyperelliptic curves over elliptic curves is that a smaller base field can be used in order to obtain the same level of security. This makes hyperelliptic curves suitable when only limited memory and computing power is available. Hyperelliptic curves are also of interest because in 2000, Gaudry, Hess and Smart [15] proposed an algorithm which reduces ECDLP over F_{2^m}, for special values of n, to the hyperelliptic curve DLP (HCDLP) over an sub field of F_{2^m}.

Table 1.1 NIST Guidelines for Public-Key Sizes with Equivalent Security Levels

Security (bits)	Symmetric encryption algorithm	Minimum size (bits) of public keys		
		DSA/DH	RSA	ECC
80	Skipjack	1024	1024	160
112	3DES	2048	2048	224
128	AES-128	3072	3072	256
192	AES-192	7680	7680	382
256	AES-256	15360	15360	512

1.2 The organization of the thesis

The rest of this thesis is organized as follows.

In Chapter 2, we first review some important background in algebra, and introduce algebraic geometry including variety, algebraic curves, and so on. We also introduce the divisors on a curve which are useful for computing Weil pairing in

elliptic curves [24]. The group on a hyperelliptic curve is also based on the divisors.

In Chapter 3, we define the hyperelliptic curves over a finite field and the additive group Jacobian associated with a hyperelliptic curve. After defining Jacobian group, we describe the Mumford representation which is used in Cantor's algorithm to compute the group operation.

In Chapter 4, we describe the index calculus algorithm to solve hyperelliptic curve discrete logarithm problems and several improvements in recent years including the ideas of reduced factor base and large primes. The double large prime variation of hyperelliptic curve index calculus is better than others, and even better than Pollard's rho method when the genus of the hyperelliptic curve is larger than 2.

In the case of genus 2 curves, Pollard's rho algorithm is faster than index calculus algorithm. In Chapter 5, we propose an algorithm for solving genus 2 HCDLP which has the same time complexity as Pollard's rho method. Several computational comparisons are given in section 5.4 to shows that our algorithm is faster than Pollard's rho method in practice.

Finally, we summarize our results and propose future work in Chapter 6.

Chapter 2

Mathematical Background

This chapter introduces some elementary mathematical background used in this thesis, including definitions and theorems in abstract algebra and algebraic geometry. If the readers are interested in more of the background, [10] and [11] give good introductions. In section 2.3, we introduce the divisor theory which is the basis of hyperelliptic curve group law. For more details on divisor theory, the reader is referred to [24][25].

2.1 Abstract algebra

Definition 2.1 (Group)

A group (G, *) is a set G with a binary operation * that satisfies the following four axioms:

- Closure: For all a, b in G, the result of a * b is also in G.
- Associativity: For all a, b and c in G, (a * b) * c = a * (b * c).
- Identity element: There exists an element e in G such that for all a in G, e*a= a*e= a.
- Inverse element: For each a in G, there exists an element b in G such that a* b= b* a = e, where e is an identity element.

Definition 2.2 (Abelian group)

A group G is said to be an abelian group (or commutative) if the operation is commutative, that is, for all a, b in G, a * b = b * a.

Definition 2.3 (Cyclic group)

A cyclic group is a group whose elements can be generated by successive composition of the group operation being applied to a single element of that group. This single element is called the generator or primitive element of the group.

Example 2.1

$<Z_5, +>$ is an additive group under the addition modulo 5. The group is cyclic since it can be generated by a single element "1", i.e. $Z_5 = <1> = \{1, 2, 3, 4, 0\}$.

Theorem 2.1 (Lagrange's theorem)

For any finite group G, the order (number of elements) of every subgroup H of G divides the order of G.

Proof:

This can be shown using the concept of left cosets of H in G. The left cosets are the equivalence classes of a certain equivalence relation on G and therefore form a partition of G. If we can show that all cosets of H have the same number of elements, then we are done, since H itself is a coset of H. Now, if aH and bH are two left cosets of H, we can define a map $f : aH \rightarrow bH$ by setting $f(x) = ba^{-1}x$. This map is bijective because its inverse is given by $f^{-1}(y) = ab^{-1}y$.

This proof also shows that the quotient of the orders |G| / |H| is equal to the index [G:H] (the number of left cosets of H in G). If we write this statement as $|G| = [G:H] \cdot |H|$.

Definition 2.4 (Ring)

A ring is a set R equipped with two binary operations + and · , called addition

and multiplication, such that:

- (R, +) is an abelian group with identity element 0,
- Multiplication is associative,
- Multiplication distributes over addition:
  - $a\cdot(b + c) = (a\cdot b) + (a\cdot c)$
  - $(a + b)\cdot c = (a\cdot c) + (b\cdot c)$

Definition 2.5 (Ideal)

An additive subgroup I of a ring R satisfying the properties: $rx \in I$, $xr \in I$ for $x \in I$ and $r \in R$ is an ideal.

Example 2.2

- The set of integers $\mathbb{Z}$ is a ring, and the set of even integers $2\mathbb{Z}$ is an ideal of $\mathbb{Z}$.
- The set R[x] of all polynomials in one variable x with coefficients in a ring R is a ring under polynomial addition and multiplication.

Definition 2.6 (Integral domain)

An integral domain is a commutative ring with $0 \neq 1$ such that ab = 0 implies that either a = 0 or b = 0 (the zero-product property). That is to say, it is a nontrivial ring without left or right zero divisors.

Definition 2.7 (Field)

A field (F, +, *) is defined by these properties:

- (F, +) is an abelian group with the additive identity 0.
- (F\{0}, *) is an abelian group with the multiplicative identity 1.
- The operation * is distributive over the operation +. For all a, b, c, belonging to

F, a * (b + c) = (a * b) + (a * c).

Definition 2.8 (Subfield, extension field)

A subset K of a field L is a subfield of L if K is itself a field with respect to the operations of L. L is said to be an extension field of K.

Fact 2.1 (Existence and uniqueness of finite fields)

1. If K is a finite field, then K contains p^d elements with p prime and d >= 1.

2. For every prime power order p^d, there is a unique (up to isomorphism) finite field of order p^d. It is an algebraic extension of degree d of F_p. The notation for a finite field of order q is F_q with $q = p^d$.

Definition 2.9 (Algebraic closure)

A field K is said to be algebraically closed if every polynomial $f \in K[x]$ has a zero in K. Such a polynomial splits into linear factors over K.

Fact 2.2 (Algebraic closure of F_p)

The algebraic closure $\overline{F_q}$ of a finite field F_q is given by: $\overline{F_q} = \bigcup_{k=1}^{\infty} F_{q^k}$

Lemma 2.1 (Frobenius Automorphism)

Let F_q be a finite field with $q=p^d$. Then we have:

(i) $a = a^p$ with $a \in F_p$

(ii) $(a \cdot b)^p = a^p \cdot b^p$ for $a, b \in F_q$

(iii) $(a+b)^p = a^p+b^p$ for $a, b \in F_q$

Consequently the following mapping is an automorphism:

$\sigma: F_q \rightarrow F_q$ where $\sigma(a) = a^p$ for $a \in F_q$

It is called the Frobenius Automorphism of F_q.

Proof:

(i) Since F_p^* is a cyclic group of order p-1, we have $a^{p-1} = 1$ for all $a \in F_p^*$. Thus $a^p = a$ for all $a \in F_p$.

(ii) It's true since the operation $\cdot$ is commutative.

(iii) $(a+b)^p = \sum_{i=0}^{p} \binom{p}{i} a^i b^{p-i} = a^p + b^p$

Notice that the binomial coefficients $\binom{p}{i}$ for i = 1,…,p-1 are multiples of the characteristic p and reduce to zero.

Definition 2.10 (Galois Group)

Let F_q be a field with $q=p^d$. Let σ be the Frobenius Automorphism of F_q and let $a \in F_q$. A power of σ is defined as:

$$\sigma^j(a) = a^{p^j}$$

The Galois Group is the group of all automorphisms acting on the field F_q, which leave the points of F_p invariant. It is a cyclic group of order d given by 1, σ, …, σ^{d-1}. That is the Galois Group $Gal(F_q/F_p) = \{1, \sigma, \ldots, \sigma^{d-1}\}$.

2.2 Algebraic geometry

Let $\overline{K}$ be an algebraic closed field, we can define the following terms.

Definition 2.11 (Affine n-space)

The affine n-space is the set of n-tuples called points:

$$A^n = A_{\overline{K}}^n = \{p = (x_1, \ldots, x_n) : x_i \in \overline{K}\}.$$

Definition 2.12 (Affine algebraic set)

For each subset S of $\overline{K}[x_1,...,x_n]$, define the zero-locus of S to be the set of points in A^n on which the functions in S vanish:

$$Z(S) = \{p \in A^n \mid f(p) = 0 \text{ for all } f \in S\}.$$

A subset V of A^n is called an affine algebraic set if V = Z(S) for some S.

Definition 2.13 (Affine variety)

A nonempty affine algebraic set V is called irreducible if it cannot be written as the union of two proper affine algebraic subsets. An irreducible affine algebraic set is called an affine variety.

Definition 2.14 (Ideal of an affine variety)

Given a subset V of A^n, let I(V) be the ideal of all functions vanishing on V:

$$I(V) = \left\{ f \in \overline{K}[x_1,...,x_n] \mid f(p) = 0 \text{ for all } p \in V \right\}.$$

Similarly, we can define projective variety in projective space.

Definition 2.15 (Projective n-space)

The projective n-space over $\overline{K}$, denoted $P^n_{\overline{k}}$, or simply P^n, is the set of equivalence classes of (n+1)-tuples $(x_0,...,x_n)$ of elements of $\overline{K}$, not all zero, under the equivalence relation given by $(x_0,...,x_n) \sim (\lambda x_0,...,\lambda x_n)$ for all $\lambda \in \overline{K}$, $\lambda \neq 0$.

An element of P^n is called a point. If P is a point, then any (n+1)-tuple $(x_0,...,x_n)$ in the equivalence class P is called a set of ***homogeneous coordinates*** for P.

Definition 2.16 (Homogeneous polynomial)

A polynomial $f \in \overline{K}[x_0,...,x_n]$ is a homogeneous polynomial if

$f(\lambda x_0, ...\lambda x_n) = \lambda^{\deg(f)} f(x_0, ...x_n)$.

Definition 2.17 (Homogeneous ideal)

An ideal $I \subset \overline{K}[x_0, ..., x_n]$ is a homogeneous ideal if it is generated by homogeneous polynomials.

The homogeneity of the polynomial ensures that this construction is well-defined.

Definition 2.18 (Projective algebraic set, projective variety)

For each set S of homogeneous polynomials, define the zero-locus of S to be the set of points in P^n on which the functions in S vanish:

$$Z(S) = \{p \in P^n \mid f(p) = 0 \text{ for all } f \in S\}.$$

A subset V of P^n is called a *projective algebraic set* if V = Z(S) for some S. An irreducible projective algebraic set is called a *projective variety*.

Definition 2.19 (Ideal of a projective variety)

Given a subset V of P^n, let I(V) be the ideal generated by all homogeneous polynomials vanishing on V: $I(V) = \left\{ f \in \overline{K}[x_0, ..., x_n] \mid f(p) = 0 \text{ for all } p \in V \right\}$

Definition 2.20 (Algebraic curve)

An algebraic curve over a field K is an equation *f*(x, y) =0, where *f*(x, y) is a polynomial in x and y with coefficients in K. A point on an algebraic curve is simply a solution of the equation of the curve. A K-rational point is a point (x, y) on the curve, where x and y are in the field K.

Definition 2.21 (Points at infinity)

Each affine space can be identified with a unique projective space. The points in P^n, which are not defined in the corresponding affine space A^n are called *points at infinity*.

For example, an affine variety C(I) is called an algebraic curve when I(C) consists of one polynomial in two variables which by definition of variety is irreducible. We will use C as the notation of an affine variety for which is an algebraic curve.

Definition 2.22 (Coordinate ring, polynomial function)

The *coordinate ring* of C is the quotient ring given by: $\overline{K}[C] = \frac{\overline{K}[x,y]}{I(C)}$.

Similarly the coordinate ring of C/K is the quotient ring given by: $K[C] = \frac{K[x,y]}{I(C)}$.

An element of $\overline{K}[C]$ is called a *polynomial function* on C.

Definition 2.23 (Function field, rational function)

The *function field* $\overline{K}(C)$ is given by the field of fractions of $\overline{K}[C]$: $\overline{K}(C) = \left\{ \frac{G}{H} \mid G, H \in \overline{K}[C] \right\}$. Similarly the function field K(C) is given by the field of fractions of K[C]. An element of $\overline{K}(C)$ is called a *rational function* on C.

Definition 2.24 (Zero, pole)

Let $f \in \overline{K}(C)$ be a non-zero rational function and P∈C. Then f is said to be defined at P if there exists a representation f = g/h, where g, h∈$\overline{K}[C]$, with h(P)≠0. If $f(P) = 0$, then f is said to have a zero at P. If f is not defined at P then f

is said to have a pole at P. In this case we write $f(P) = \infty$.

Definition 2.25 (Order)

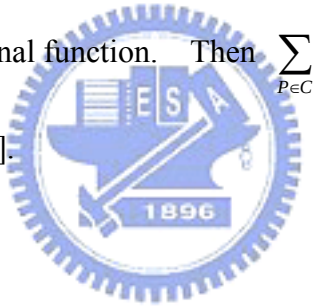
The order of a polynomial function $g \in \overline{K}[C]$ at a point $P \in C$ is the intersection multiplicity at that point and denoted by order $ord_P(g)$. Notice that P is a zero of g if and only if $ord_P(g) > 0$, and P is a pole of g if and only if $ord_P(g) < 0$.

The order of a rational function $f = g/h \in \overline{K}(C)$ at a point $P \in C$ is defined as $ord_P(f) = ord_P(g) - ord_P(h)$.

Theorem 2.2

Let $f \in \overline{K}(C)$ be a rational function. Then $\sum_{P \in C} ord_P(f) = 0$.

This proof can be found in [24].



2.3 Divisor theory

Divisors are useful for keeping track of the zeros and poles of a rational function. In this section we give the basic definitions and properties of divisors. For simplicity, we are working in an algebraic closure $\overline{K}$. Later we will give the definitions over a finite field K in chapter 3.

Definition 2.26 (Divisor, degree, order, support)

A *divisor* D is a formal sum of points in C: $D = \sum_{P \in C} m_P P$, $m_P \in \mathbb{Z}$, where only a finite number of m_P is non-zero.

The *degree* of D is the integer $\deg(D) = \sum_{P \in C} m_P$.

The *order* of D at P is the integer $ord_P(D) = m_P$.

The *support* of D is the set supp (D) $= \{P \in C \mid m_P \neq 0\}$.

Definition 2.27 (Divisor group)

The set of all divisors, denoted by **D**, forms an additive group under the addition rule:

$$\sum_{P \in C} m_P P + \sum_{P \in C} n_P P = \sum_{P \in C} (m_P + n_P) P .$$

The set of all divisors of degree 0, denoted $\mathbf{D^0}$, is a subgroup of **D**.

Definition 2.28 (Gcd of divisors)

Let $D_1 = \sum_{P \in C} m_P P$, $D_2 = \sum_{P \in C} n_P P$ be two divisors. The greatest common divisor of D_1 and D_2 is defined to be $\gcd(D_1, D_2) = \sum_{P \in C} \min(m_P, n_P) P - \left(\sum_{P \in C} \min(m_P, n_P) \right) \infty$.

(Note that gcd (D_1, D_2) $\in \mathbf{D^0}$.)

Definition 2.29 (Principal divisor)

Let $R \in \overline{K}(C)$. The divisor of R is called a *principal divisor* $div(R) = \sum_{P \in C} ord_P(R) P$. Theorem 2.2.16 shows that the divisor of a rational function is indeed a finite formal sum and has degree 0.

Definition 2.30 (Principal divisor group)

The group of principal divisors is a subgroup of $\mathbf{D^0}$ and is defined by:

$P = P(C) = \{div(R) \mid R \in \overline{K}(C)\}$. We have that $P \subset D^0 \subset D$.

Definition 2.31 (Jacobian)

The Jacobian of the curve C is defined by the quotient group:

$\mathbf{J = J(C) = D^0/P}$.

If $D_1, D_2 \in \mathbf{D^0}$ then we write $D_1 \sim D_2$ *if* $D_1 - D_2 \in \mathbf{P}$; D_1 and D_2 are said to be equivalent divisors.

Example 2.3 (Elliptic curve)

Consider the following algebraic curve in affine space:

$$I(C_R): \quad f(x,y) = y^2 - (x^3 - x + 1) \quad \text{in} \quad \mathbb{R}[x,y]$$

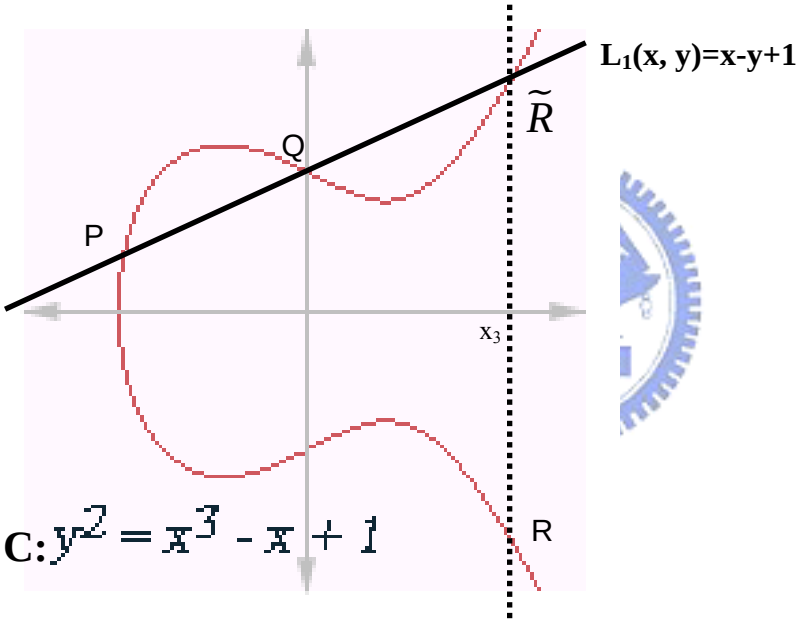


Figure 2.1 An elliptic curve C and rational function L_1 over $\mathbb{R}$

The algebraic closure of $\mathbb{R}$ is the field of the complex numbers; we still denote it as $\overline{K}$.

The affine variety over $\overline{K}$ is given by $I(C) = \{(x,y) \mid x, y \in \overline{K}, f(x,y) = 0\}$.

The coordinate ring of C is given by the quotient ring :

$$\overline{K}[C] = \overline{K}[x,y] / \left(y^2 - (x^3 - x + 1)\right).$$

The function field of C is given by: $\overline{K}(C) = \left\{ \frac{g}{h} \mid g, h \in \overline{K}[C] \right\}$.

The line through the point P, Q is a rational function given by $L_1(x, y)$.

$$div(L_1) = P + Q + \widetilde{R} - 3\infty = (P - \infty) + (Q - \infty) + (\widetilde{R} - \infty).$$

The vertical line through R and $\widetilde{R}$ is a rational function given by $L_2(x, y)=x-x_3$.

$$div(L_2) = R + \widetilde{R} - 2\infty = (R - \infty) + (\widetilde{R} - \infty).$$

Let $D_1 = P - \infty$, $D_2 = Q - \infty$ and $D_3 = R - \infty$ in the Jacobian $\mathbf{J} = \mathbf{D^0/P}$, we have :

$$\begin{aligned} &(D_1 + D_2) - D_3 \\ &= (P - \infty) + (Q - \infty) - (R - \infty) \\ &= div(L_1) - div(L_2) \\ &= div\left(\frac{L_1}{L_2}\right) \in P \end{aligned}$$

Hence $D_1+D_2 \sim D_3$, the Jacobian group law is the same as point addition on an elliptic curve.

Chapter 3

Hyperelliptic Curves

Hyperelliptic curve is a kind of algebraic curve, and elliptic curve is a special case of hyperelliptic curve. In chapter 2, we defined the function field of an algebraic curve. The Jacobian is the group of degree zero divisors modulo principal divisors, i.e. the quotient group $\mathbf{J = D^0/P}$ over an algebraic closed field $\overline{K}$. Since the implementation of arithmetic on a hyperelliptic curve works with the base field K, we need to know the definitions over K.

Let C be a hyperelliptic curve defined over a finite field K. Let P = (x, y) $\in$ C, and let σ be an automorphism of $\overline{K}$ over K which means σ is an isomorphism from $\overline{K}$ to itself and σ(x) = x for all x $\in$ K. Then $P^\sigma := (x^\sigma, y^\sigma)$ is also a point on C, and $\infty^\sigma = \infty$.

Definition 3.1 (Field of definition of a divisor)

A divisor $D = \sum m_P P$ is said to be defined over K if $D^\sigma := \sum m_P P^\sigma$ is equal to D for all automorphisms of $\overline{K}$ over K.

Notice that the set of all automorphisms of $\overline{K}$ over K is the Galois Group $Gal\left(\overline{K}/K\right)$ defined in Definition 2.10 (Galois Group).

If a divisor D is defined over K, it does not mean that each point in the *support* of D is a K-rational point. A principal divisor is defined over K if and only if it is a divisor of a rational function which has coefficients in K. The set $\mathbf{J_C(K)}$ of all

divisors defined over K in **J** is a subgroup of **J**.

Since each element of the Jacobian is a coset, we need a unique representation for the divisors in the Jacobian. Such divisors exist and are called reduced divisor, which is introduced in section 3.2. In section 3.3, we introduce the Mumford's representation [27]: a reduced divisor can be represented by the gcd of two polynomials a(x) and y- b(x). The points associated to the corresponding divisor are the roots of both a(x) and y- b(x). These two polynomials can also be seen as ideals modulo *principal ideals*. The equivalence classes are called *ideal classes*. Adding divisors in the Jacobian is the same as composing ideals. Cantor's algorithm [2] can efficiently compute the group operation of two divisors in the Jacobian.

3.1 Definitions and properties

We use K to denote a field and $\overline{K}$ to denote the algebraic closure of K in this chapter.

Definition 3.2 (Hyperelliptic curve)

A hyperelliptic curve of genus g over K is an equation of the form C: $y^2 + h(x)y = f(x)$ in K[x, y], where deg(h(x)) ≦ g, deg(f(x)) = 2g+1, f(x) is a monic polynomial, and the integer g ≧ 1. A hyperelliptic curve C should be **non-singular**, that is, there are no solutions (x, y)$\in \overline{K} \times \overline{K}$ on curve C which satisfy both partial derivative equations $2y + h(x) = 0$ and $h'(x)y - f'(x) = 0$.

Definition 3.3 (K-rational points)

The set $C(K) = \{(x, y) \mid x, y \in K, y^2 + h(x)y = f(x)\} \cup \{\infty\}$ is called the set of

K-rational points on C. The point ∞ is called the *point at infinity*.

Definition 3.4 (Opposite, special and ordinary points)

For P=(x, y) $\in$C the *opposite* of P is the point $\widetilde{P} = (x, -y - h(x))$. If P=$\widetilde{P}$ then it is called *special* point, otherwise it is called *ordinary*. The opposite of the point at infinity ∞ is defined as $\widetilde{\infty} = \infty$, hence is a special point.

Lemma 3.1

Let C: $y^2 + h(x)y = f(x)$ be a hyperelliptic curve defined over K. If the characteristic of K is odd, then C can be transformed to the form $y^2 = f_1(x)$ where $f_1(x)$ has no repeated roots in $\overline{K}$.

Proof:

Under the change of variables x→x, y→(y-h(x)/2), the equation of C is transformed to $(y - \frac{h(x)}{2})^2 + h(x)(y - \frac{h(x)}{2}) = f(x)$, which simplifies to $y^2 = f(x) + \frac{h(x)^2}{4} = f_1(x)$.

Since C is a hyperelliptic curve, there is no point (x, y)$\in \overline{K} \times \overline{K}$ satisfying $y^2=f_1(x)$, $2y=0$, and $f_1'(x)=0$. Therefore, $f_1(x)$ has no repeated roots.

Lemma 3.2

The polynomial $F(x, y) = y^2 + h(x)y - f(x)$ is irreducible over $\overline{K}$.

Proof:

Suppose F(x, y) is reducible over $\overline{K}$, then $F(x, y) = (y - a(x))(y + b(x))$ $= y^2 + (b(x) - a(x))y - a(x)b(x)$ for some a, b$\in \overline{K}[x]$. But deg(a(x)b(x)) = deg(f(x)) = 2g+1 and deg(a(x)+b(x)) = deg(h(x)) $\leqq$g which is impossible.

3.2 Reduced divisors

We defined the Jacobian of curves in chapter 2, and with the definitions in section 3.1, we know that the Jacobian of a hyperelliptic curve C is **J = D^0/P**. Note that two divisors D_1 and D_2 in **J** are said to be equivalent if they are in the same equivalence class, i.e. D_1-$D_2 \in \mathbf{P}$, denoted by $D_1 \sim D_2$. In the following we introduce reduced divisor to uniquely represent the divisors in the same equivalence class of **J**.

Definition 3.5 (Semi-reduced divisor)

A semi-reduced divisor is a degree zero divisor of the form $D = \sum_{P \in C \setminus \infty} m_P P - \sum_{P \in C \setminus \infty} m_P \infty$ with the following properties:

(i) $m_P > 0$,

(ii) if $P \neq \tilde{P}$ and $m_P > 0$ then $m_{\tilde{P}} = 0$,

(iii) if $P = \tilde{P}$ and $m_P > 0$ then $m_P = 1$.

Definition 3.6 (Reduced divisor)

Let $D = \sum_{P \in C \setminus \infty} m_P P - \sum_{P \in C \setminus \infty} m_P \infty$ be a semi-reduced divisor. If $\sum_{P \in C \setminus \infty} m_P \leq genus$ then D is called a reduced divisor.

Lemma 3.3

For each divisor $D \in D^0$ there exists a semi-reduced divisor $D_1 \in D^0$ such that $D \sim D_1$.

Proof:

Let $D = \sum_{P \in C \setminus \infty} m_P P - m\infty$. Let ($C_1$, C_2, C_3) be the partition of the support of D, such that $C_1 = \{P \in C \setminus \infty \mid P \neq \widetilde{P}, m_P \geq m_{\widetilde{P}}\}$, $C_2 = \{\widetilde{P} \in C \setminus \infty \mid P \neq \widetilde{P}, m_P \geq m_{\widetilde{P}}\}$, and $C_1 = \{P \in C \setminus \infty \mid P = \widetilde{P}\}$. Then $D = \sum_{P \in C_1} m_P P + \sum_{P \in C_2} m_P P + \sum_{P \in C_3} m_P P - m\infty$. Let

$$D_1 = D - \sum_{P=(x_P, y_P) \in C_2} m_P \cdot div(x - x_P) - \sum_{P=(x_P, y_P) \in C_3} \left\lfloor \frac{m_P}{2} \right\rfloor \cdot div(x - x_P)$$

$$= \sum_{P \in C_1} (m_P - m_{\widetilde{P}})P + \sum_{P \in C_3} (m_P - 2 \cdot \left\lfloor \frac{m_P}{2} \right\rfloor)P - m_1\infty \quad \text{for some } m_1 \in Z.$$

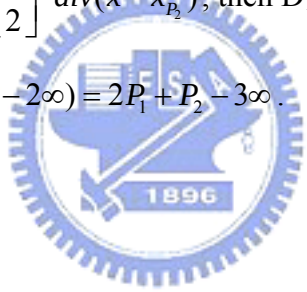
Hence $D_1 \sim D$ and D_1 is semi-reduced.

For example, let $D = 6P_1 + 4\widetilde{P_1} + 3P_2 - 13\infty \in D^0$ where $P_1 \neq \widetilde{P_1}$ and $P_2 = \widetilde{P_2}$. Then $C_1 = \{P_1\}$, $C_2 = \{\widetilde{P_1}\}$, and $C_3 = \{P_2\}$.

Let $D_1 = D - 4 \cdot div(x - x_{P_1}) - \left\lfloor \frac{3}{2} \right\rfloor \cdot div(x - x_{P_2})$, then $D_1 \sim D$.

$D_1 = D - 4(P_1 + \widetilde{P_1} - 2\infty) - (2P_2 - 2\infty) = 2P_1 + P_2 - 3\infty$.

Hence D_1 is semi-reduced.



Theorem 3.1 [25]

For each divisor $D \in D^0$ there exists a unique reduced divisor D_1 such that $D \sim D_1$.

3.3 Representation

When we implement a hyperelliptic curve cryptosystem, we work over a finite field K. In the following, we introduce the computational representation of reduced divisors of the Jacobian defined over K, which is so-called Mumford representation [27].

Fact 3.1 (Mumford representation)

For a hyperelliptic curve C: $y^2 + h(x)y = f(y)$ in $K[x, y]$, and $D = \sum_{P_i=(x_i,y_i)\in C} m_i P_i - (\sum m_i)\infty$ be a semi-reduced divisor, we can use two polynomials a(x), b(x) $\in K[x]$ to uniquely represent D. Let $a(x) = \prod (x - x_i)^{m_i}$.

Let b(x) be the unique polynomial satisfying:

(i) $\deg_x(b) < \deg_x(a)$,

(ii) $b(x_i) = y_i$ for all i which $m_i \neq 0$,

(iii) a(x) divides $(b(x)^2 + b(x)h(x) - f(x))$.

Then D = gcd(div(a(x)), div(b(x)-y)); we usually simplify the notation as div(a, b).

If D=div(a, b) is a *reduced divisor*, then $\deg_x(a) = \sum m_i \leq genus$.

The zero divisor, the identity of $\mathbf{J_C(K)}$, is represented by *div(1, 0)*. The opposite of a divisor div(a, b) is given by *div(a, -h-b)*, which is also called *involution*. This means div(a, b) + div(a, -h-b) ~ div(1, 0) under the Jacobian group law.

Fact 3.2 (Hasse-Weil Bound)

Let C be a hyperelliptic curve of genus g defined over F_q. Then the bound of the order of $\mathbf{J_C(F_q)}$ is given by:

$$\left(\sqrt{q} - 1\right)^{2g} \leq \# J_C(F_q) \leq \left(\sqrt{q} + 1\right)^{2g},$$

and the number of F_q-rational points is:

$$q + 1 - 2g\sqrt{q} \leq \#C(F_q) \leq q + 1 + 2g\sqrt{q}.$$

As a result, we know that $\# J_C(F_q) \approx q^g$ and $\#C(F_q) \approx q$.

3.4 Group law

By using Mumford representation described in the previous section, Cantor's algorithm [2] can compute the group operation of $\mathbf{J_C(K)}$ efficiently.

Algorithm 3.1 (Cantor's algorithm)

Input: Reduced divisors $D_1 = \mathrm{div}(a_1, b_1)$ and $D_2=\mathrm{div}(a_2, b_2) \in \mathbf{J_C(K)}$.

Output: The reduced divisor $D_3 = \mathrm{div}(a_3, b_3)$ sucht that $D_3 \sim D_1+D_2$.

Phase 1: (Composition)

1. Compute $d_1 = \gcd(a_1, a_2) = e_1a_1 + e_2a_2$

2. Compute $d = \gcd(d_1, b_1+b_2+h) = c_1d_1 + c_2(b_1+b_2+h)$

3. Let $s_1 = c_1e_1$, $s_2 = c_1e_2$, and $s_3 = c_2$, so that

$$d = s_1a_1 + s_2a_2 + s_3(b_1+b_2+h)$$

4. Set $a = \frac{a_1a_2}{d^2}$

and $b = \frac{s_1a_1b_2 + s_2a_2b_1 + s_3(b_1b_2 + f)}{d} \mod a$

Phase 2: (Reduction)

5. Set $a' = \frac{f - bh - b^2}{a}$

and $b' = (-h - b) \mod a$

6. If $\deg(a') > g$ then set $a \leftarrow a'$, $b \leftarrow b'$ and go to step 5.

7. Make a' monic, and output $(a_3, b_3) = (a', b')$.

In Cantor's algorithm, the composition phase gives a semi-reduced divisor $\mathrm{div}(a, b) \sim D_1+D_2$, and the reduction phase reduces a semi-reduced divisor to the unique reduced divisor.

Here is an example that illustrates how Cantor's algorithm works.

Example 3.1

Let C: $y^2+y = x^5+1$ be a hyperelliptic curve of genus 2 over finite field F_2. Given $D_1=\text{div}(x+1, 0)$ and $D_2=\text{div}(x^2+1, x) \in \mathbf{J_C(F_2)}$.

$d = \gcd(a_1, a_2, b_1+b_2+h) = \gcd(x+1, x^2+1, x+1) = x+1 \Rightarrow s_1=1, s_2=s_3=0.$

$$a = \frac{a_1 a_2}{d^2} = \frac{(x+1)(x^2+1)}{(x+1)^2} = x+1.$$

$$b = \frac{s_1 a_1 b_2 + s_2 a_2 b_1 + s_3 (b_1 b_2 + f)}{d} \mod a = \frac{(x+1)x}{x+1} \mod (x+1)=1.$$

Since $\deg(a)=1 \leqq 2$, the divisor div(a, b) is already reduced.

Then, we have $D_1+ D_2 = \text{div}(x+1, 0) + \text{div}(x^2+1, x) = \text{div}(x+1, 1)$.

In recent years, several researchers have derived the explicit formulas for small genus hyperelliptic curves from Cantor's algorithm. They investigate what can be the input of Cantor's algorithm and proceed in considering these different cases. With careful analysis, some redundant field operations can be omitted in explicit formulas. For example, Lange [22] presents explicit formulas for the group law of genus 2 hyperelliptic curves, and the most common case in the addition of two reduced divisor requires 1 inversion, 12 multiplications, and 2 squarings. The explicit formulas for genus 3 hyperelliptic curves can be found in [17]. When genus becomes higher than 4, the explicit formulas is getting too complicated and may not be possibly derived by hand.

3.5 Hyperelliptic curve discrete log problem (HCDLP)

The security of several cryptosystems is related to the difficulty of computing discrete logarithms modulo a large prime number p; i.e. given two numbers (g mod p) and (g^x mod p), it seems to be infeasible to compute x when p is large enough. Instead of using the DLP modulo a large prime p as the basis of cryptographic protocols, one can consider the DLP in an arbitrary group that admits an efficient

element representation and group law.

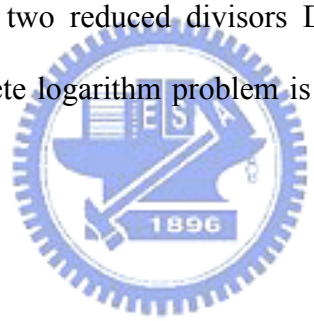
Definition 3.7 (DLP)

Let G be a finite cyclic group G= <g> of order n, and given an element $h \in G$.. The discrete logarithm problem is to find the integer $x \in [0, n-1]$, such that $g^x=h$.

Since the Jacobian of a hyperelliptic curve is also a finite abelian group, based on the difficulty of the DLP, it can be designed for cryptographic use.

Definition 3.8 (HCDLP)

Let C be a hyperelliptic curve over a finite field F_q and $\mathbf{J_C(F_q)}$ its Jacobian with order # $\mathbf{J_C(F_q)}$ = n. Given two reduced divisors D_1, $D_2 \in \mathbf{J_C(F_q)}$ and $D_2 \in <D_1>$. The hyperelliptic curve discrete logarithm problem is to find the integer $\lambda \in [0, n-1]$, such that $\lambda D_1=D_2$.



Example 3.2

Consider the genus 2 hyperelliptic curve: C: $y^2 = x^5 + 2x^4 + 1$ in $F_3[x, y]$. The partial derivatives are $2x^4 + 2x^3=0$ and $2y=0$. Since there are no points in $\overline{F} \times \overline{F}$ which satisfy C and the partial derivatives, the hyperelliptic curve is non-singular.

Although the divisors are defined over F_3, the points in the support of a divisor are in F_{3^2}.

The finite field $F_{3^2} \cong F_3[x]/(x^2+1) = \{0, 1+i, 2i, 1+2i, 2, 2+2i, i, 2+i, 1\}$.

The F_{3^2}-rational points are $P_1 = (0, 1)$, $P_2 = (1, 2)$, $P_3 = (1, 1)$, $P_4 = (0, 2)$,
$P_5 = (2+i, 2+2i)$, $P_6 = (2+2i, 2+i)$, $P_7 = (i, 2+i)$, $P_8 = (2i, 2+2i)$,
$P_9 = (i, 1+2i)$, $P_{10} = (2i, 1+i)$, $P_{11} = (2+i, 1+i)$, $P_{12} = (2+2i, 1+2i)$, ∞.

The order of Jacobian $\#J_C(F_3) = 17$.

Let $D_1 = \text{div}(x^2, 1)$. We can use D_1 as the generator of the group, and use

Cantor's algorithm to generate the group elements.

1 $D_1 = \mathrm{div}(x^2, 1)$ $= P_1 + P_1 - 2\infty$

2 $D_1 = \mathrm{div}(x+2, 2)$ $= P_2 - \infty$

3 $D_1 = \mathrm{div}(x^2+2x+2, 2x+1)$ $= P_5 + P_6 - 2\infty$

4 $D_1 = \mathrm{div}(x^2+x+1, x+1)$ $= P_2 + P_2 - 2\infty$

5 $D_1 = \mathrm{div}(x^2+1, x+1)$ $= P_9 + P_{10} - 2\infty$

6 $D_1 = \mathrm{div}(x^2+2x, 2x+2)$ $= P_3 + P_4 - 2\infty$

7 $D_1 = \mathrm{div}(x^2+2x, 1)$ $= P_1 + P_3 - 2\infty$

8 $D_1 = \mathrm{div}(x, 2)$ $= P_4 - \infty$

9 $D_1 = \mathrm{div}(x, 1)$ $= P_1 - \infty$

10 $D_1 = \mathrm{div}(x^2+2x, 2)$ $= P_2 + P_4 - 2\infty$

11 $D_1 = \mathrm{div}(x^2+2x, x+1)$ $= P_1 + P_2 - 2\infty$

12 $D_1 = \mathrm{div}(x^2+1, x+2)$ $= P_7 + P_8 - 2\infty$

13 $D_1 = \mathrm{div}(x^2+x+1, 2x+2)$ $= P_3 + P_3 - 2\infty$

14 $D_1 = \mathrm{div}(x^2+2x+2, x+2)$ $= P_{11} + P_{12} - 2\infty$

15 $D_1 = \mathrm{div}(x^2+2, 1)$ $= P_3 - \infty$

16 $D_1 = \mathrm{div}(x^2, 2)$ $= P_4 + P_4 - 2\infty$

17 $D_1 = \mathrm{div}(1, 0)$

Chapter 4

Algorithms for HCDLP

4.1 Introduction

The best known algorithm for solving the DLP in generic groups is Pollard's rho algorithm. Pollard's algorithm has an exponential expected running time of $\sqrt{\frac{\pi n}{2}}$ group operations and negligible storage requirements. In order to prevent such square-root attacks, the group order n must have a large prime factor. There are faster algorithms for the DLP than Pollard's rho method. The most powerful is the index calculus method which yields subexponential-time algorithms for the DLP in some groups.

The first *subexponential-time* algorithm to compute discrete logarithms over hyperelliptic curves of large genus is introduced by Adleman, DeMassais and Huang [1] in 1994. This algorithm was rather theoretical, and some improvements on it were done by other researchers. Flassenberg and Paulus [9] implemented a sieve version of this algorithm, but the consequence for cryptographical applications is not clear. Enge [6] improved the original algorithm and gave a precise evaluation of the running time, but did not implement his ideas. Muller, Stein and Thiel [26] extended the resultsto the real quadratic congruence function fields. Smart and Galbraith [12] also gave some ideas in the context of the Weil descent, following ideas of Frey; they dealt with general curves (not hyperelliptic). We will not discuss those in details but list them as references.

When the index calculus algorithm is applied on the small genus HCDLP, even the fastest variation is not faster than Pollard's rho method for the genus less than 3. Hence the use of hyperelliptic curves in public-key cryptography appears as an alternative to the use of elliptic curves, with the advantage that it can be used in a smaller base field for the same level of security. In order to analyze the security of such systems, we need to know how the index calculus method works for solving small genus HCDLP.

In 2000, Gaudry [13] first presented a variation of index calculus attack for a hyperelliptic curve of genus g over F_q that could solve the HCDLP in time $O(q^2)$. And Harley [13] improved this algorithm with reduced factor base such that HCDLP can be solved in time $O\left(q^{2-\frac{2}{g+1}}\right)$. Furthermore, Thériault improved it by using the almost-smooth divisor which contains exactly one large prime. Theriault's algorithm [32] works in time $O\left(q^{2-\frac{4}{2g+1}}\right)$.

By considering double large prime, the time complexity of hyperelliptic index calculus algorithm can be reduced to $O\left(q^{2-\frac{2}{g}}\right)$. This idea was proposed independently by Gaudry et al. [16] and Naogo [28] in 2004. They used different tricks to handle large primes, but got the same time complexity. We discuss these variations of index calculus algorithm for small genus HCDLP in section 4.2.

However, the double large prime variation can not be applied on genus 2 hyperelliptic curves. We propose an algorithm that can solve the genus 2 HCDLP with time complexity O(q) in Chapter 5 which can be comparable to Pollard's rho method. **Table 4.1** shows the comparison between these algorithms described above. Our algorithm has the same time complexity as Pollard's rho method but smaller

hiding constant term. We also have detailed analysis in Chapter 5.

Table 4.1 Time complexity of algorithms solving HCDLP

Genus g	2	3	4	5	6
Pollard's rho	q	$q^{\frac{3}{2}}$	q^2	$q^{\frac{5}{2}}$	q^3
Original index calculus	q^2	q^2	q^2	q^2	q^2
with reduced factor base	$q^{\frac{4}{3}}$	$q^{\frac{3}{2}}$	$q^{\frac{8}{5}}$	$q^{\frac{5}{3}}$	$q^{\frac{12}{7}}$
with single large prime	$q^{\frac{6}{5}}$	$q^{\frac{10}{7}}$	$q^{\frac{14}{9}}$	$q^{\frac{18}{11}}$	$q^{\frac{22}{13}}$
with double large prime	—	$q^{\frac{4}{3}}$	$q^{\frac{3}{2}}$	$q^{\frac{8}{5}}$	$q^{\frac{5}{3}}$
Our algorithm	q	—	—	—	—

4.2 Index calculus algorithm for small genus HCDLP

A reduced divisor in the Jacobian $\mathbf{J_C(K)}$ is represented by two polynomials (a, b), and the factorization of a as polynomial in K[x] is compatible with the Jacobian group law. This is the key stone for defining a smooth divisor and then the index calculus algorithm.

Fact 4.1 (Factorization)

Let C be a hyperelliptic curve over a finite field F_q. Let D=div(a, b) be a reduced divisor in $\mathbf{J_C(F_q)}$. Factor a(x) as $a(x)=\prod a_i(x)$ where $a_i(x)$ are

irreducible factors of a(x) in $F_q[x]$. Let $b_i(x) = b(x) \pmod{a_i(x)}$.

Then $D_i = \text{div}(a_i, b_i)$ is a reduced divisor and $D = \sum D_i$ in $\mathbf{J_C(F_q)}$.

Remark 4.1

To factor polynomials over finite fields we can use the Cantor-Zassenhaus algorithm, which is invented by D. Cantor and Hans Zassenhaus in 1981 [3]. It is currently implemented in many well-known computer algebra systems.

With this result in **Fact 4.1**, a reduced divisor can be rewritten as the sum of reduced divisors of smaller $\deg(a_i)$, and $\deg(a) = \sum \deg(a_i)$. If the a-polynomial of a reduced divisor D is irreducible then it can not be rewritten as their decomposition. We call them primes in $\mathbf{J_C(F_q)}$.

Definition 4.1 (Prime)

A reduced divisor $D=\text{div}(a, b) \in \mathbf{J_C(F_q)}$ is said to be prime if the polynomial a is irreducible in $F_q[x]$.

Definition 4.2 (B-smooth)

Let B be an integer. A divisor is said to be B-smooth if all the prime divisors in its factorization of a-polynomial have degree at most B. When B= 1, a 1-smooth divisor will be a divisor for which the polynomial a splits completely over F_q.

We give a sketch of the index calculus algorithm in the following. Several improvements described in this section are based on this algorithm.

Algorithm 4.1 Hyperelliptic index calculus algorithm
Input: A divisor D_1 in $\mathbf{J_C(F_q)}$ with know order n = ord(D_1), and a divisor $D_2 \in \langle D_1 \rangle$. Output: An integer λ sucht that $D_2=\lambda D_1$.
1. Fix smoothness bound B and construct the factor base F. 2. While not enough relations have been found do: Pick a random element $R=\alpha D_1+\beta D_2$. If R is smooth, record the corresponding relation. 3. Solve the linear algebra system over Z_n. 4. Return λ.

The factor base F contains all the prime reduced divisors which a-polynomial has degree at most B: $F = \{D \in J_C(F_q) : D = div(a,b) \text{ is prime}, \deg(a) \le B\}$. For convenience, we use g_i for i=1,2,…,#F to denote the element in F. To find all the prime divisors in F, it suffices to test all the monic polynomial a(x) of degree at most B, checking if it is irreducible and if there exists a polynomial b(x) such that div(a, b) $\in \mathbf{J_C(F_q)}$.

While searching the smooth relations in step 2, a naive way to select a random element $R=\alpha D_1+\beta D_2$ is costly: two integers α and β are randomly chosen in [0, n-1] and then two scalar multiplications have to be done. It costs O(log n) group operations. We can use a pseudo random walk instead, so that each new random element R costs just one group operation.

Let $R_0 = \alpha_0 D_1 + \beta_0 D_2$ be the starting point of the walk where α_0 and β_0 are random integers in [0, n-1]. For j from 1 to r, we compute random divisors $T_j = a_j D_1 + b_j D_2$. The walk R_{i+1} will then be given by adding one of the T_j to R_i.

The index $j \in [1, r]$ is given by a hush function H evaluated at R_i. In other words, $R_{i+1}=R_i+T_j$ where $j=H(R_i) \in [1, r]$, and $\alpha_{i+1}=\alpha_i + a_j$, $\beta_{i+1} = \beta_i + b_j$. Once the initialization is finished, we can compute a new pseudo-random element R_{i+1} at the cost of one addition in the Jacobian. Practical experiments suggest that by taking r= 20 the pseudo random walk behaves almost like a purely random walk.

For each R_i of the random walk, test its smoothness by factoring the a-polynomial of R_i. If all its irreducible factors have degree at most B (then it is smooth), express it on the factor base; otherwise, throw it away. Thus we collect a subsequence of the sequence (R_i) where all the divisors are smooth. We denote this subsequence by (S_k) with kth smooth element $S_k=\alpha_k D_1+\beta_k D_2$. Hence we can put the result of this computation in a matrix M, each column representing an element of the factor base, and each row being a reduced divisor S_k expressed on the basis: for a row k, we have $S_k = \sum_{1 \le i \le \#F} m_{ki} g_i = \alpha_k D_1 + \beta_k D_2$, where $M = (m_{ki})$. We collect #F + 1 rows in order to have a $(\#F+1) \times \#F$ matrix. Thus the kernel of the transpose of M is of dimension at least 1.

Using linear algebra, we find a non-zero vector (γ_k) of this kernel, which corresponds to a relation between the S_k's. So that $\sum_k \gamma_k S_k = 0 = \left(\sum_k \gamma_k \alpha_k\right) D_1 + \left(\sum_k \gamma_k \beta_k\right) D_2$, and then $\lambda = -\frac{\sum_k \gamma_k \alpha_k}{\sum_k \gamma_k \beta_k} (\mathrm{mod}\, n)$. The discrete logarithm is now found with high probability, because the denominator is zero with probability $\frac{1}{n}$.

In this algorithm, there are two crucial points: one is to search enough smooth relations, and another is to solve the large linear system. In the matrix obtained in the algorithm, each row is a smooth divisor written as sum of at most g elements of the factor base. Hence the matrix is very sparse, and we have at most g terms in

each row. For such a sparse matrix, Lanczos's [21] or Wiedemann's [33][5] algorithm can be used, in order to get a solution in time quadratic in the number of rows, instead of cubic by Gaussian elimination.

We know that the index calculus algorithm can solve HCDLP in a subexponential time $O\left(L_{q^g}\left(\frac{1}{2},\sqrt{2}\right)\right)$ when $g \gg \log q$ [1], where $L_N(\alpha,c) = \exp\left(c(\log N)^{\alpha}(\log\log N)^{1-\alpha}\right)$. When the genus is relatively small (say at most 9), the theoretical optimal smoothness bound $B = \left\lceil \log_q L_{q^g}\left(\frac{1}{2},\sqrt{2}\right)\right\rceil$ which tends to 0. In this case, B= 1 is the best choice. The first index calculus algorithm for hyperelliptic curve of small genus was proposed by Gaudry in 2000. We summarize in the following algorithm.

Algorithm 4.2 Index calculus algorithm for small genus HCDLP
Input: A hyperelliptic curve C of small genus g over F_q, a divisor D_1 in $\mathbf{J_C(F_q)}$ with know order n = ord(D_1), and a divisor $D_2 \in \langle D_1 \rangle$. Output: An integer λ sucht that $D_2=\lambda D_1$.
1. /* Build the factor base F */ For each monic irreducible polynomial a_i over F_q of degree 1, try to find b_i such that div(a_i, b_i) is a divisor of the curve. If there is a solution, store g_i =div(a_i, b_i) in F. 2. /* Initialization of the random walk */ For j from 1 to 20, select a_j and b_j at random in [0, n-1], and compute $T_j := a_jD_1 + b_jD_2$. Select α_0 and β_0 at random in [0, n-1] and compute $R_0 := \alpha_0D_1 + \beta_0D_2$.

Set k to 1.

3. /* Main loop */

(a) /* Look for a smooth divisor */

Compute $j := H(R_0)$, $R_0 := R_0 + T_j$, $\alpha_0 := \alpha_0 + a_j \bmod n$, and $\beta_0 := \beta_0 + b_j \bmod n$.

Repeat this step until R_0 is a smooth divisor.

(b) /* Express R_0 on the factor base F */

Factor $a_0(u)$ over F_q, and determine the positions of the factors in the basis G..

Store the result as a row $R_k = \sum m_{ki} g_i$ of a matrix $M = (m_{ki})$.

Store the coefficients $\alpha_k = \alpha_0$ and $\beta_k = \beta_0$.

If $k < \#F + 1$, then set $k := k + 1$, and return to step 3.a.

4. /* Linear algebra */

Find a non-zero vector (γ_k) of the kernel of the transpose of the matrix M.

The computation can be done in Z_n.

5. /* Solution */

Return $\lambda = -\dfrac{\sum_k \gamma_k \alpha_k}{\sum_k \gamma_k \beta_k} (\bmod n)$.

Lemma 4.1

The proportion of smooth divisors in the Jacobian of a curve of genus g over F_q tends to $\dfrac{1}{g!}$.

Proof:

By the Hasse-Weil bound, $\#F = \#C(F_q) = O(q)$ and $\#\mathbf{J_C(F_q)} = O(q^g)$. The smooth divisors can be written as the sum of at most g points in $C(F_q)$, hence we have about $\dfrac{q^g}{g!}$ smooth divisors in $\mathbf{J_C(F_q)}$. The proportion is $\dfrac{1}{g!}$.

In step 1, we need to perform q times a resolution of an equation of degree 2 over

F_q. Step 2 requires a constant number of Jacobian operations. Step 3 is a loop of O(q) times to find enough smooth relations. In step 4, this linear algebra step consists in finding a vector of the kernel in a sparse matrix of size O(q), and of weight O(gq); the coefficient are in Z_n. Hence Lanczos's algorithm provides a solution with cost $O(gq^2)$. This last step requires only O(q) multiplications modulo n, and one inversion. When q is large, we can regard g and logq as small constant. Then the complexity of this algorithm is $O(q^2)$.

Theorem 4.1 [13]

Let C be a hyperelliptic curve of genus g over the finite field F_q. If q>g! then the discrete logarithms in $\mathbf{J_C(F_q)}$ can be computed in expected time $O\left(g^3 q^{2+\varepsilon}\right)$.

Example 4.1

Given a genus 2 hyperelliptic curve C: $y^2 = x^5 + 2x^4 + 1$ over F_3. This curve is also used as an example in Example 3.2. Let $D_1 = \text{div}(x^2, 1)$ with $\text{ord}(D_1) = 17$, and $D_2 = \text{div}(x^2+1, x+2) \in \langle D_1 \rangle$. We can use the index calculus algorithm described in Algorithm 4.2 to find an integer λ such that $D_2 = \lambda D_1$.

1. Construct factor base

   $F = \{g_1=\text{div}(x, 1), g_2=\text{div}(x+2, 2), g_3=\text{div}(x+2, 1), g_4=\text{div}(x, 2)\}$.

2. Initialize the pseudo-random walk:

   $T_1 = 2D_1 + 10D_2 = \text{div}(x^2+2x+2, 2x+1)$

   $T_2 = 13D_1 + 5D_2 = \text{div}(x^2+1, x+1)$

   $T_3 = 3D_1 + 7D_2 = \text{div}(x+2, 2)$

3. Search enough smooth relations by using a pseudo random walk:

   $R_0 = 1D_1+1D_2 = \text{div}(x^2+x+1, 2x+2) = 2g_3$.

   $R_1 = R_0+T_2 = 14D_1+6D_2 = \text{div}(x^2, 1) = 2g_1$.

   $R_2 = R_1+T_1 = 16D_1+16D_2 = \text{div}(x^2+x+1, x+1) = 2g_2$.

   $R_3 = R_2+T_1 = 1D_1+9D_2 = \text{div}(x^2+2x, 1) = g_1+g_3$.

   $R_4 = R_3+T_3 = 4D_1+16D_2 = \text{div}(x, 1) = g_1$.

   If R_i is smooth we can store it in a matrix M, otherwise discard it.

These smooth relations are stored in a matrix M:

α_i	1	14	16	1	4
β_i	1	6	16	9	16
g_1	0	2	0	1	1
g_2	0	0	2	0	0
g_3	2	0	0	1	0
g_4	0	0	0	0	0

Matrix M

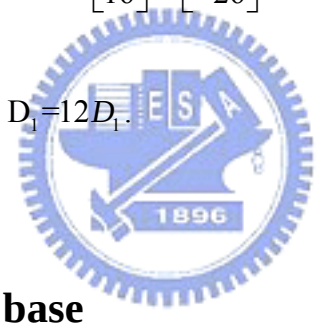
4. When there is enough(#F+1 = 5) smooth relations, we can find a non-trivial kernel r of M, such that rM=0. We have r =(0, 1, 0, 0, -2)^T.

$$r \cdot M = 0 + \begin{bmatrix} 2 \\ 0 \\ 0 \\ 0 \end{bmatrix} + 0 + 0 - 2 \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\Rightarrow r \cdot \begin{bmatrix} \alpha_i \\ \beta_i \end{bmatrix} = 0 + \begin{bmatrix} 14 \\ 6 \end{bmatrix} + 0 + 0 - 2 \cdot \begin{bmatrix} 4 \\ 16 \end{bmatrix} = \begin{bmatrix} 6 \\ -26 \end{bmatrix}$$

Hence, $6D_1 - 26D_2 = 0$,

$\Rightarrow D_2 = [6/26 \pmod{17}]\, D_1 = 12D_1$.



4.2.1 Reduced factor base

Because the running time for Gaudry's algorithm is dominated by the cost of solving the linear algebra, a natural approach to improve the algorithm is to reduce the cost of linear algebra part. Hence we need to reduce the size of the linear system, which means reducing the size of factor base. This was first introduced by Robert Harley. We can choose the factor base F with $|F|=q^r$ where r is a real number in the interval (0, 1). This increases the cost of searching relation, because it also reduces the proportion of the smooth divisors in the Jacobian. To balance the cost of the relation search and linear algebra $r \approx \frac{g}{g+1}$ is the best choice. Then, the time

complexity of the index calculus algorithm with reduced factor base is $O\left(q^{2-\frac{2}{g+1}}\right)$.

Theorem 4.2 [13]

Let C be a hyperelliptic curve of genus g over the finite field F_q. If q>g! then the discrete logarithms in $\mathbf{J_C(F_q)}$ can be computed in expected time $O\left(g^5 q^{2-\frac{2}{g+1}+\varepsilon}\right)$.

4.2.2 Single large prime variation

As the index calculus algorithm for the multiplicative group of a finite field, the hyperelliptic index calculus algorithm can be improved by using large primes.

Definition 4.3 (Large prime)

Let r be a real number such that 0<r<1. A subset S of F_q of size q^r is fixed arbitrarily. The factor base F is the set $F = \{P = (x, y) \in C(F_q) \subset J_C(F_q); x \in S\}$. The set of large primes L is the set $L = \{P \in C(F_q) \subset Jac_C(F_q)\} \setminus F$.

We have $\#F \approx q^r$ and $\#L \approx q$. The union of factor base and large primes is the set of F_q-rational points $(x_i, y_i) \in C(F_q)$ which can represent the prime divisors with $\mathrm{div}(a_i, b_i) = \mathrm{div}(x-x_i, y_i)$.

Definition 4.4 (1-almost smooth divisor)

A reduced divisor $D = \sum m_i P_i - m\infty$ is said to be 1-almost smooth if all but exactly one of the P_i's are in F and the remaining P_i is a large prime.

Definition 4.5 (2-almost smooth divisor)

A reduced divisor $D = \sum m_i P_i - m\infty$ is said to be 2-almost smooth if all but exactly two of the P_i's are in F and the remaining P_i's are two large primes.

Simple combinatorial arguments give good estimates for the probabilities of obtaining almost smooth divisors in the relation search.

Lemma 4.2

The probability for a random divisor to be smooth is approximately $\frac{q^{g(r-1)}}{g!}$. The probability for a random divisor to be 1-almost smooth is approximately $\frac{q^{(g-1)(r-1)}}{(g-1)!}$. The probability for a random divisor to be 2-almost smooth is approximately $\frac{q^{(g-2)(r-1)}}{2(g-2)!}$.

We now consider the single large prime variation of the index calculus algorithm. In order to take advantage of the high number of 1-almost smooth divisors, we must find pairs of these divisors with the same large prime. For example, given two 1-almost smooth divisors $D_1 = \sum_{P_i \in F} m_i P_i + n_1 Q - *\infty$, $D_2 = \sum_{P_i \in F} m_i P_i + n_2 Q - *\infty$ where Q is a large prime, then we can obtain a smooth divisor by computing $n_2 D_1 - n_1 D_2$.

The following algorithm shows how this method can be applied in the relation search of the original index calculus algorithm.

Algorithm 4.4.5: Searching relation with single large prime
Input: A hyperelliptic curve C of small genus g over F_q,
a divisor D_1 in $\mathbf{J_C(F_q)}$ with know order n = ord(D_1),

a divisor $D_2 \in \langle D_1 \rangle$,

a factor base F, and the set of large primes L.

Output: A system of k smooth divisors of the form $R_i = \alpha_i D_1 + \beta_i D_2$.

1. /* Initialization of the random walk */

For j from 1 to 20, select a_j and b_j at random in [0, n-1], and compute

$T_j := a_j D_1 + b_j D_2$.

Select α and β at random in [0, n-1] and compute $R := \alpha D_1 + \beta D_2$.

P←{}

i←1

2. /* Main loop */

While i≦k do{

R←R+T_j for some randomly chosen j, update α and β.

Decompose R into prime divisors

If R is smooth then

R_i←R

i←i+1

If R is 1-almost smooth with a large prime Q then

If Q is already in P then

Obtain a smooth divisor R by cancelling the large prime Q

R_i←R

i←i+1

else (Q is not in P)

Add Q to the set P with the associated relation R

}

Return {R_1, R_2, …, R_k}

According to Theriault's analysis, by choosing the factor base F such that $|F| = O\left(g^2 q^{\left(g-\frac{1}{2}\right)/\left(g+\frac{1}{2}\right)+\varepsilon}\right)$, we get the following result:

Theorem 4.3 [32]

Let C be a hyperelliptic curve of genus g over the finite field F_q. If q>g! then the discrete logarithms in $\mathbf{J_C(F_q)}$ can be computed in expected time $O\left(g^5 q^{2-\frac{4}{2g+1}+\varepsilon}\right)$.

Example 4.2

Given the same HCDLP as in **Example 4.1**.

Let C: $y^2 = x^5 + 2x^4 + 1$ over F_3.

$D_1 = \text{div}(x^2, 1)$ with $\text{ord}(D_1) = 17$, and $D_2 = \text{div}(x^2+1, x+2) \in \langle D_1 \rangle$. We want to find an integer λ such that $D_2 = \lambda D_1$.

1. Construct factor base $F = \{g_1=\text{div}(x, 1), g_2=\text{div}(x+2, 2), g_3=\text{div}(x+2, 1)\}$, and the set of large primes is $\{g_4=\text{div}(x, 2)\}$.
2. Initialize the pseudo-random walk:
   $T_1 = 2D_1 + 10D_2 = \text{div}(x^2+2x+2, 2x+1)$
   $T_2 = 13D_1 + 5D_2 = \text{div}(x^2+1, x+1)$
   $T_3 = 3D_1 + 7D_2 = \text{div}(x+2, 2)$
3. Search enough smooth relations by using a pseudo random walk:
   $R_0 = 16D_1+8D_2 = \text{div}(x^2+2x, 2) = g_2+g_4$.
   $R_1 = R_0+T_1 = 1D_1+1D_2 = \text{div}(x^2+x+1, 2x+2) = 2g_3$.
   $R_2 = R_1+T_2 = 14D_1+6D_2 = \text{div}(x^2,1) = 2g_1$.
   $R_3 = R_2+T_2 = 10D_1+11D_2 = \text{div}(x^2+2x, 2x+2) = g_3+g_4$.
   $R_4 = R_3+T_2 = 6D_1+16D_2 = \text{div}(x^2+2x, x+1) = g_1+g_2$.

R_0 and R_3 are 1-almost smooth relations with the same large prime g_4. We can calculate a smooth relation $R' = R_0-R_3 = 6D_1-3D_2 = g_2-g_3$.

These smooth relations are stored in a matrix M:

α_i	1	14	6	6
β_i	1	6	-3	16
g_1	0	2	0	1
g_2	0	0	1	1
g_3	2	0	-1	0

Matrix M

4. When there is enough(#F+1 = 4) smooth relations, we can find a non-trivial kernel r of M, such that rM=0. We have r =$(1, 1, 2, -2)^T$.

$$r \cdot M = \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix} + \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix} + 2 \cdot \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix} - 2 \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\Rightarrow r \cdot \begin{bmatrix} \alpha_i \\ \beta_i \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 14 \\ 6 \end{bmatrix} + 2 \cdot \begin{bmatrix} 6 \\ -3 \end{bmatrix} - 2 \cdot \begin{bmatrix} 6 \\ 16 \end{bmatrix} = \begin{bmatrix} 15 \\ -31 \end{bmatrix}$$

Hence, $15D_1-31D_2 = 0$,
$\Rightarrow D_2 = [15/31 \pmod{17}]\, D_1 = 12D_1$.

4.2.3 Double large prime variation

Since 1-almost smooth divisors can be used to produce relations so much faster, it is natural to also consider 2-almost smooth divisors. By the definition of 2-almost smooth divisor, the smallest genus g of a hyperelliptic curve is 3 such that a reduced divisor which is 2-almost smooth is of the form $D=P+Q_1+Q_2-3\infty$ where P is in factor base and Q_i are large primes. Here is an example to cancel the large primes.

Example 4.3

Let C be a hyperelliptic curve of genus g=3. $D_1=P_1+Q_1+Q_2-3\infty$, $D_2=P_2+Q_2+Q_3-3\infty$, and $D_3=P_3+Q_3+Q_1-3\infty$ where P_i are in the factor base and Q_i are large primes. We can cancel the large primes by multiplying the divisors by a

relative constant and adding them together. In this example, the constants are 1 or -1. Hence $D_1 - D_2 + D_3 = P_1 - P_2 + P_3 + 2Q_1 - 3\infty$ is a 1-almost smooth divisor. If we have another 1-almost smooth divisor $D_4=P_4+Q_1-2\infty$ then we can get a smooth divisor by $D_1 - D_2 + D_3 - 2D_4 = P_1 - P_2 + P_3 - 2P_4 + \infty$.

To manipulate the chain of almost smooth divisors, Gaudry and Thome introduce the graph of large prime relations (LP-graph, in short). LP-graph is an undirected acyclic graph with 1+#L vertices, corresponding to the elements of the set of large primes L and the special vertex 1. All edges of the LP-graph are labeled with a relation. An edge between vertex 1 and vertex Q_i represents a 1-almost smooth divisor with the large prime Q_i, and an edge between vertex Q_i and Q_j represents a 2-almost smooth divisor with these two large primes.

Algorithm 4.3 Searching relation with double large primes

Input: A hyperelliptic curve C of small genus $g \geqq 3$ over F_q,
a divisor D_1 in $\mathbf{J_C(F_q)}$ with know order $n = ord(D_1)$,
a divisor $D_2 \in \langle D_1 \rangle$,
a factor base F, and the set of large primes L.

Output: a system of k smooth divisors of the form $R_i = \alpha_i D_1 + \beta_i D_2$.

1. /* Initialization of the random walk */
   For j from 1 to 20, select a_j and b_j at random in [0, n-1], and compute
   $T_j := a_j D_1 + b_j D_2$.
   Select α and β at random in [0, n-1] and compute $R := \alpha D_1 + \beta D_2$.
   G←empty graph
   i←1
2. /* Main loop */
   While $i \leqq k$ do{
      R←R+T_j for some randomly chosen j, update α and β.
      Decompose R into prime divisors

If R is smooth then

$R_i \leftarrow R$

$i \leftarrow i+1$

If R is 1-almost smooth with a large prime Q then

If there is a path from vertex 1 to Q then

Obtain a smooth divisor R by cancelling the large primes in the path

$R_i \leftarrow R$

$i \leftarrow i+1$

Leave G unchanged

else (edge(1, Q) would not create a cycle in G)

Add edge(1, Q) to G with the associated relation R

If R is 2-almost smooth with large prime Q_1, Q_2 then

If edge(Q_1, Q_2) would create a cycle containing vertex 1 in G then

Use the relations in the cycle to cancel the large primes and

then obtain a new smooth divisor R.

$R_i \leftarrow R$

$i \leftarrow i+1$

Leave G unchanged

If edge(Q_1, Q_2) would create a cycle not containing vertex 1 in G then

Use the relations in the cycle to cancel the large primes other than Q_1

If Q_1 is also canceled then a new smooth divisor R is obtained

$R_i \leftarrow R$

$i \leftarrow i+1$

Leave G unchanged

Else (the new divisor R is 1-almost smooth divisor with Q_1)

Add edge(1, Q_1) to G with the associated relation R

Else ((Q_1, Q_2) is not connected in G)

Add edge(Q_1, Q_2) to G with the associated relation R

}

Return {R_1, R_2, …, R_k}

To test if adding an edge would create a cycle, we can use the union-and-find algorithm to find if these two vertices of the edge are in the same set (connected

component).

Theorem 4.4 [16]

Let C be a hyperelliptic curve of genus g≧3 over the finite field F_q. If q>g! then the discrete logarithms in $\mathbf{J_C(F_q)}$ can be computed in expected time $O\left(g^5 q^{2-\frac{2}{g}+\varepsilon}\right)$.

4.3 Computational comparison

4.3.1 Solving large sparse linear system

The last step in the index calculus algorithm is to solve a large sparse linear system over finite field. We implemented Lanczos's algorithm to do this work. Here is the Lanczos's algorithm to solve the system $Ax = w$ for a column n-vector x, where A is a n×n matrix and w is a column n-vector.

Algorithm 4.4 Lanczos's algorithm
Input: A n×n matrix A and a column n-vector w Output: A column n-vector x for the system $Ax = w$.
1. $w_0 = w$, $v_1 = Aw_0$, $w_1 = v_1 - \frac{(v_1, v_1)}{(w_0, v_1)} w_0$. 2. i = 1 While ($w_i$, Aw_i) ≠ 0 do

$v_{i+1} = Aw_i$,

$w_{i+1} = v_{i+1} - \frac{(v_{i+1}, v_{i+1})}{(w_i, v_{i+1})} w_i - \frac{(v_{i+1}, v_i)}{(w_{i-1}, v_i)} w_{i-1}$,

i = i+1.

3. If $w_i = 0$, then

$x = \sum_{j=0}^{i-1} \frac{(w_j, w)}{(w_j, v_{j+1})} w_j$ is a solution.

In **Algorithm 4.4**, the notation (,) denote the inner product of two vectors.

In general, the systems we need to solve are not symmetric, and are of the form Bx = u, where B is m×n matrix, m≧n, x is an unknown column n-vector, and u is a given column m-vector. Suppose we need to solve the system Bx = u over field K. Let D be a m×m diagonal matrix with the diagonal elements randomly selected from K\{0}, and let

$A = B^T D^2 B$,

$w = B^T D^2 u$.

We can expect that with high probability a solution to the system $Ax = w$ is a solution to the system $Bx = u$.

4.3.2 Curve selection

To select a suitable hyperelliptic curve, we need to check if the order of Jacobian has a large prime factor in order to avoid the square root attacks. Koblitz first described a method of calculating the number of points on the Jacobian of a hyperelliptic curve of genus 2 and of small characteristic, by using zeta functions. Sakai and Sakurai [30] improved the method by proposing a point counting method for curves of small characteristic but of arbitrary genus.

Algorithm 4.5 Sakai and Sakurai method

Input: A hyperelliptic curve C of genus 2 over the field F_{q^n}.

Output: $\#J_C\left(F_{q^n}\right)$.

1. Determine N_r, the number of points on the curve over F_{q^r} for r=1,…,g.
2. Determine the coefficients of $L_{F_q}(t)=\sum_{i=0}^{2g} a_i t^i$ in the following way:
   (a) $a_0 = 1$
   (b) for $1 \le i \le g$: $a_i = \left(\sum_{k=1}^{i}\left(N_k - \left(q^k+1\right)\right)a_{i-k}\right)/i.$
   (c) for $g+1 \le i \le 2g$: $a_i = q^{i-g}a_{2g-i}$.
3. Compute $L_{F_{q^n}}(1)=\prod_{k=1}^{n} L_{F_q}\left(\zeta^k\right)$, where ζ runs over the n-th root of unity.

Return $\#J_C(F_q)=L_{F_{q^n}}(1)$.

Note that it should be easy to count N_1, …, N_g if F_q is small, so this algorithm is only suitable for fields of small characteristic.

We have implemented the Sakai and Sakurai method to select the suitable Jacobian of a hyperelliptic curve having an order which containing a large prime factor.

Example 4.4 lists the test data we use to test the index calculus algorithms for solving HCDLP in section 4.3.3 and section 5.4.

Example 4.4

Genus 2 hyperelliptic curves:

(a) C: $y^2+y=x^5+x^4+x^3$ over $F_{2^{11}}$, $\#J_C(F_{2^{11}})=4196353=7\times 599479$.

(b) C: $y^2+(x^2+x+1)y=x^5+x^4$ over $F_{2^{13}}$,

$\#J_C(F_{2^{13}})=66695006=2\times 7\times 4763929$.

(c) C: $y^2+y=x^5+x^3$ over $F_{2^{17}}$, $\#J_C(F_{2^{13}})=17247109633=13\times 1326700741$.

(d) C: $y^2+(x^2+x+1)y=x^5+x^4$ over $F_{2^{19}}$,

$\#J_C(F_{2^{19}})=274720225346=2\times 7\times 19622873239$.

Genus 3 hyperelliptic curves:

(e) C: $y^2+(x^3+x^2+1)y=x^7+1$ over $F_{2^{11}}$,

$\#J_C(F_{2^{11}})=8589762730=2\times 5\times 858976273$.

(f) C: $y^2+(x^3+x^2+1)y=x^7+1$ over $F_{2^{13}}$,

$\#J_C(F_{2^{13}})=549756909530=2\times 5\times 131\times 419661763$.

(g) C: $y^2+(x^3+x^2+1)y=x^7+x^6+x^5$ over $F_{2^{17}}$,

$\#J_C(F_{2^{17}})=2255872542702704=2^4\times 140992033918919$.

(h) C: $y^2+\left(x^3+x^2+1\right)y=x^7+1$ over $F_{2^{19}}$,

$\#J_C(F_{2^{19}})$=144115188252574570 $=2\times 5\times 14411518825257457$.

Genus 4 hyperelliptic curves:

(i) C: $y^2+x^3y=x^9+x$ over $F_{2^{11}}$,

$\#J_C(F_{2^{11}})=18566518893488=2^4\times 1160407430843$.

(j) C: $y^2+x^3y=x^9+x$ over $F_{2^{13}}$,

$\#J_C(F_{2^{13}})=4546690000751824=2^4\times 284168125046989$.

4.3.3 Comparisons

We have implemented the index calculus algorithms and several variations described in section 4.2 including original index calculus, index calculus with reduced factor base, index calculus with single large prime, index calculus with double large primes. In order to implement these algorithms we use the C++ library NTL [31] to manipulate the operations over finite field. NTL (Number Theory Library) is a high-performance, portable C++ library providing data structures and algorithms for manipulating signed, arbitrary length integers, and for vectors, matrices, and polynomials over the integers and over finite fields.

We ran our programs on the computer with 1800 MHz CPU and 1G ram to generate the results in **Table 4.2**.

Table 4.2 Running time (seconds) of hyperelliptic index calculus

Genus g	2			3			4	
Field size q = $\|F_q\|$	2^{11}	2^{13}	2^{15}	2^{11}	2^{13}	2^{15}	2^{11}	2^{13}
Original index calculus	68	3760	>3days	110	5261	>3days	1136	10321
with reduced factor base	6	34	403	93	891	10595	830	6374
with 1 large prime	2	9	18	22	533	665	248	2677
with 2 large primes	—	—	—	17	301	458	191	1813

From **Table 4.2** we can realize the following facts.

When the original index calculus is applied to small genus HCDLP, using a relative large factor base reduce the time to obtain a smooth relation but result in a large linear system which becomes dominating the running time. By using a reduced factor base to balance the search time and the time of solving linear system, the index calculus algorithm with reduced factor base solves HCDLP in a much shorter time. And the large prime variations can further improve the index calculus algorithm.

Chapter 5

A Fast Algorithm for Genus 2 HCDLP

5.1 Introduction

For genus 2 hyperelliptic curves, the index calculus algorithm is asymptotically slower than Pollard's rho method. In this chapter, we present a faster algorithm for solving genus 2 HCDLP. A comparison of the time complexity can be found in **Table 4.1**. The bottleneck of the index calculus algorithms is due to its linear algebra part. Hence the idea of our algorithm is to use a graph method to find the relation of D_1 and D_2 such that $\left(\sum_k \gamma_k \alpha_k\right) D_1 + \left(\sum_k \gamma_k \beta_k\right) D_2 = 0$ without the linear algebra part.

We choose the factor base as all the prime divisors with degree of a-polynomial being 1, which can be constructed by finding all the rational points on C in the base field. For a genus g=2 hyperelliptic curve C over F_q, if a reduced divisor is smooth then it can be represented by the sum of at most 2 points in $C(F_q)$. By **Lemma 4.1**, the probability to get a smooth divisor is $\frac{1}{g!} = \frac{1}{2}$.

Example 5.1 gives examples of all the cases that would appear in our algorithm.

Example 5.1

Let C be a hyperelliptic curve of genus 2 over F_q. Let P_i be the points of $C(F_q)$ and $R_i = \alpha_i D_1 + \beta_i D_2 \in \mathbf{J_C(F_q)}$.

(a) Let $R_1= P_1+P_2-2\infty$, $R_2= P_2+P_3-2\infty$, $R_3= P_3+P_4-2\infty$.

Then we can get a relation of P_1 and P_4 by $R_1-R_2+R_3= P_1+ P_4-2\infty$.

(b) Let $R_1= P_1+P_2-2\infty$, $R_2= P_2+P_3-2\infty$, $R_3= P_3+P_4-2\infty$, $R_4= P_4+P_1-2\infty$.

Then $R_1-R_2+R_3-R_4 = 0$

(c) Let $R_1= P_1+P_2-2\infty$, $R_2= P_2+P_3-2\infty$, $R_3= P_3+P_1-2\infty$.

Then $R_1-R_2+R_3=2P_1-2\infty$,

$R_1+R_2-R_3=2P_2-2\infty$,

$-R_1+R_2+R_3=2P_3-2\infty$,

In this case, we can not get a relation $\sum \gamma_i R_i = 0$, but we can get relations of any one of the points.

(d) Let $R_1= P_1+P_2-2\infty$, $R_2= P_2+P_3-2\infty$, $R_3= P_3+P_1-2\infty$, $R_4= P_2+P_4-2\infty$, $R_5= P_4+P_5-2\infty$, $R_6= P_5+P_2-2\infty$.

Then $(R_2-R_3+R_1)- (R_4-R_5+R_6)=(2P_2-2\infty)- (2P_2-2\infty)=0$.

(e) Let $R_1= P_1+P_2-2\infty$, $R_2= P_2+P_3-2\infty$, $R_3= P_3+P_1-2\infty$, $R_4= P_3 -\infty$.

Then $-R_1+R_2+R_3-2R_4 = 0$

The following figure shows that we can use the graph for finding cycles to get a relation $\sum \gamma_i R_i = 0$.

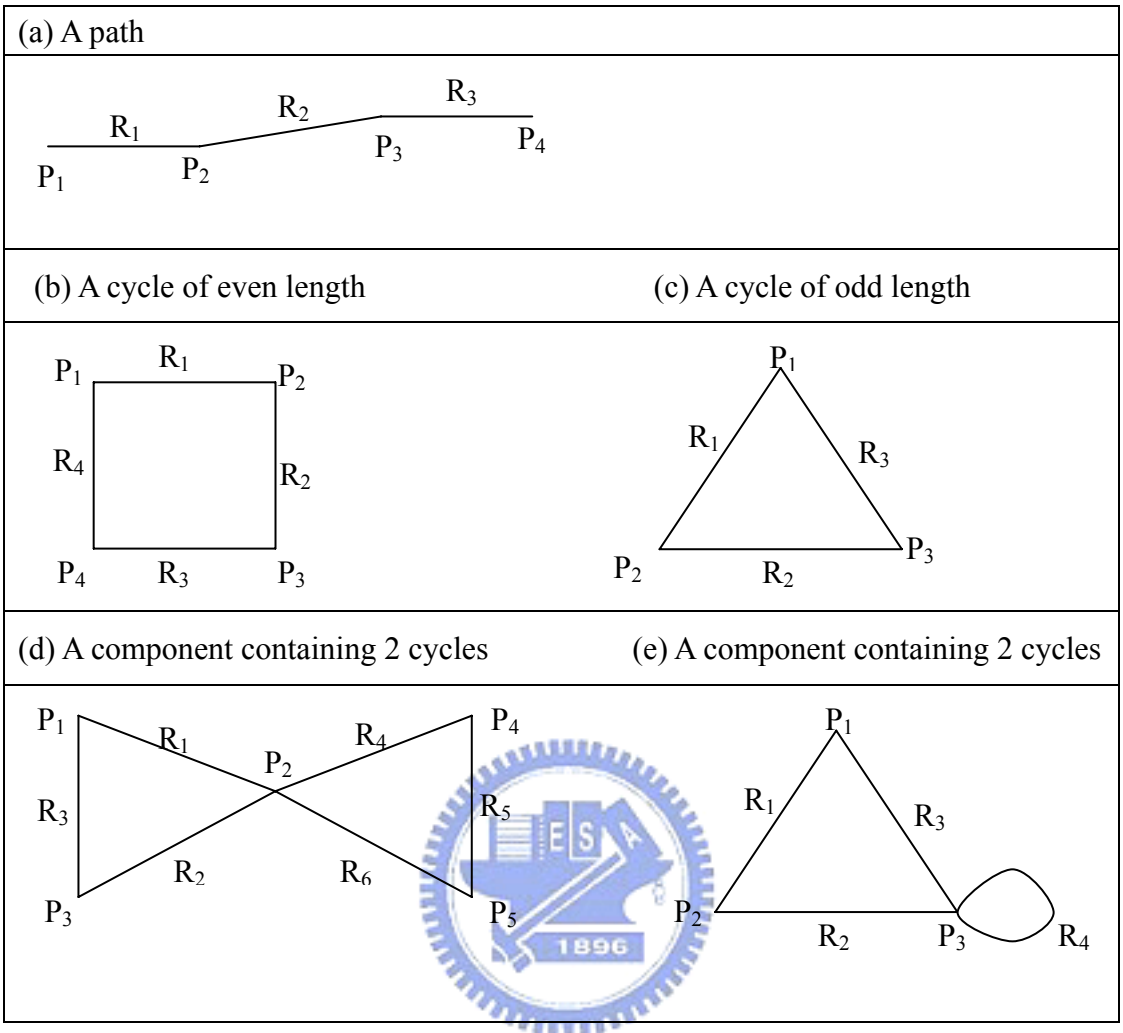


Figure 5.1 Possible sub-graphs appear in our algorithm

From the example above, we can realize some facts:

1. Case (a): If there is a path from vertex P_i to P_j then we can compute a relation for P_i and P_j.
2. Case (b): If there is a cycle of even length then we can compute a relation R such that $R = \sum \gamma_i R_i = 0$ for some γ_i.
3. Case (c): If there is a cycle of odd length then we can compute a relation of any one of the points.
4. Case (d), (e): If there is a connected component containing 2 odd length cycles then we can compute a relation R such that $R = \sum \gamma_i R_i = 0$ for some γ_i.

By regarding these edges in the graph as a smooth relation found in the relation search, it isn't hard to imagine that a new algorithm for solving genus 2 HCDLP can be designed with the graph.

5.2 The algorithm

Our algorithm first uses a pseudo random walk to create random reduced divisors of the form $R_i=\alpha_i D_1+\beta_i D_2$. Then, create a graph G with |F| vertices corresponding to the elements in the factor base F, each edge specifying a relation written as the sum of the points. Initially the graph G contains no edges. If R_i is smooth then write R_i as the sum of at most 2 points in $C(F_q)$, and then add the corresponding edge between these two points in the edge. Notice that if R_i is written as $R_i=cP_j-c\infty$ where c=1 or 2 and some $P_j \in C(F_q)$ then it is an edge of self-loop of the point P_j.

The data structure of the graph can be implemented as an array to represent trees with a union-find algorithm. In other words, we only need to record the parent node of each element in the array. To test if adding an edge (P_i, P_j) would create a cycle, we can traverse the trees from vertices P_i and P_j to see if they have the same root. If adding an edge would create an even length cycle then we get a relation $R=\sum \gamma_i R_i=\left(\sum \gamma_i \alpha_i\right)D_1+\left(\sum \gamma_i \beta_i\right)D_2=0$ for some γ_i such that the discrete logarithm of $D_2=\lambda D_1$ can be computed as $\lambda=-\frac{\sum \gamma_i \alpha_i}{\sum \gamma_i \beta_i}$. If adding an edge would create an odd length cycle then we can compute a relation $R=cP_i-c\infty$ where P_i is the root of the tree as the case (c) in **Example 5.1**. Hence we can store such information of odd length cycles (including self-loop) in the roots of the trees without creating

cycles in the graph G. If later we have another odd length cycle within the same tree then we can compute a relation $S=dP_i-d\infty$. With the information of the root P_i, the relations R and S, we can compute dR-cS=0 which implies the discrete logarithm.

Here is our algorithm in detail.

Algorithm 5.1 A faster algorithm for genus 2 HCDLP

Input: A hyperelliptic curve C of small genus g=2 over F_q,

a divisor D_1 in $\mathbf{J_C(F_q)}$ with know order n = ord(D_1),

and a divisor $D_2 \in <D_1>$.

Output: An integer λ sucht that $D_2=\lambda D_1$.

1. /* Build the factor base F */

For each $x_i \in F_q$, solve $v^2+h(x_i)v=f(x_i)$ to find $y_i \in F_q$ such that (x_i, y_i) in $C(F_q)$, and store $P_i=(x_i, y_i)$ in F.

2. /* Initialization of the random walk */

For j from 1 to 20, select a_j and b_j at random in [0, n-1], and compute

$T_j := a_jD_1 + b_jD_2$.

Select α and β at random in [0, n-1] and compute $R := \alpha D_1 + \beta D_2$.

G←empty graph

3. /* Main loop */

While G contains no even length cycles

or no component with 2 odd length cycles do

3.1 R←R+T_j for some randomly chosen j, update α and β.

3.2 If R is smooth and $R=cP_i-c\infty$ for some P_i in F, c=1 or 2

Use the relations in the path from P_i to the root of the tree containing P_i to get a relation of the root.

If there already exists a relation of the root then go to step 4.

3.3 If R is smooth and $R=P_i+P_j-2\infty$ for some P_i and P_j in F

Traverse the trees from P_i and P_j to find the roots P_{ri} and P_{rj} respectively.

3.3.1 If $P_{ri}\neq P_{rj}$ then combine these two trees

Use the relations in the path $P_{ri}\rightarrow P_i\rightarrow P_j\rightarrow P_{rj}$ to get a relation R' of P_{ri} and P_{rj}. Combine these 2 trees by adding an edge (P_{ri}, P_{rj}) and making P_{ri} as the parent node of P_{rj}. If there is a relation of P_{rj} (self-loop of P_{rj}), we also update it as a relation of P_{ri}.

If there already exists a relation of P_{ri} then go to step 4.

3.3.2 If $P_{ri}=P_{rj}$ then a cycle is found

If the cycle is of even length then go to step 4.

Else (the cycle is of odd length, store as self-loop of the root)

Use the relations in the cycle $P_{ri}\rightarrow P_i\rightarrow P_j\rightarrow P_{rj}$ to obtain a relation of P_{ri}.

If there already exists a relation of P_{ri} then go to step 4.

4. Obtain a relation of $\left(\sum \gamma_i\alpha_i\right)D_1+\left(\sum \gamma_i\beta_i\right)D_2=0$ by using the relations in an even length cycle or 2 self-loops of the same point.

Return $\lambda=-\dfrac{\sum \gamma_i\alpha_i}{\sum \gamma_i\beta_i}$ mod n.

To implement this algorithm we can use an array of #F=O(q) elements. Each element in the array contains a point P_i in $C(F_q)$ and a link to the parent node P_j with associated relation of the form $R=\alpha D_1+\beta D_2(=P_i+P_j-2\infty)$. The link is nil before such a relation appears in the pseudo random walk. Hence this algorithm requires O(q) storage space.

5.3 Time complexity

In order to analyze the time complexity of this algorithm, we refer to Flajolet, Knuth and Pittel's work [8], which provides comprehensive knowledge of the cycle appearance in random graphs. We quote some of their results in [8].

Definition 5.1 (Uniform model)

The uniform model is a procedure to enrich an initially empty graph on the vertices {1,2,…,n}. At each step we generate an ordered pair <x, y>, where x and y are uniformly distributed between 1 and n, and all n^2 pairs are equally likely. The (undirected edge) x — y is then added to the graph. In this way we obtain a multi-graph, which may have duplicate edges or self-loops x — x.

A bicyclic component in a graph is a component with more than one cycle.

Corollary 5.1 (Expected time) [8]

In the uniform model, the first cycle appears at the expected time $m \approx \frac{n}{3}$ steps. And at this time, the expected cycle length is of order $n^{\frac{1}{6}}$, and the size of the component containing the first cycle will be $\theta\left(n^{\frac{1}{2}}\right)$. The waiting time for the first bicyclic component is approximately $\frac{n}{2}$.

The graph constructed in our algorithm can be viewed as the uniform model with

|F|=O(q) vertices. At each step of pseudo random walk, the relation $R=\alpha D_1+\beta D_2$ is smooth with probability $\frac{1}{2}$. In other words, it is half chance to add an edge into the graph at each step. By **Corollary 5.1**, the first bicyclic component will appear in the graph after about $\frac{q}{2}$ edges have been added. This requires about q steps of the pseudo random walk. Hence, we conclude our algorithm solving the genus 2 HCDLP in expected time of O(q) Jacobian operations.

A practical comparison between Pollard's method and our algorithm is given in section 5.4.

5.4 Computational comparison

In this section, we implement our algorithm for solving genus 2 HCDLP, and use the implementation of Pollard's rho algorithm by Niels Lubbes [23] to be the comparison. We execute both programs on the same computer to generate the following results. The comparison between our algorithm and Pollard's rho algorithm are showed in **Table 5.1**, and the results are averages from 10 times running the tests.

Table 5.1 Comparison between Pollard's rho and our algorithm

genus		2			
Field size $q = \|F_q\|$		2^{11}	2^{13}	2^{17}	2^{19}
Pollard's rho	Average time (sec)	1.238	5.502	113.391	827.459
	Average iterations	923.4	2642.8	50239.5	236119.6
	Average number of useless collisions	1.7	0.7	1.3	2.2
Our algorithm	Average time (sec)	0.236	1.018	17.394	80.809
	Average iterations	699.4	2350.4	40222	137832
	Average number of smooth divisors	351.4	1169.2	20222.3	74338.8
	Graph size	1024	4071	65792	261993
	Average number of cycles	2.1	2.2	2.9	2.9

As we can see in **Table 5.1**, the average number of iterations in our algorithm needed for solving genus 2 HCDLP is less than the average number of iterations in Pollard's rho algorithm, and the running time of our algorithm is also less than the running time of Pollard's algorithm. For example, in the case of base field $GF(2^{19})$, Pollard's rho algorithm takes 827.459 seconds to run 236119.6 iterations in average for solving the given HCDLP, and it meets 2.2 useless collisions before the solution is found. While running our algorithm in the same case, it takes only 80.809 seconds to solve the given HCDLP. After 137832 iterations in average there are 74338.8 smooth divisors which can be added in the graph, and then average 2.9 cycles are found. The rate of $\frac{74338.8}{137832} \approx 0.539$ is about a half chance to get a smooth divisor as in **Lemma 4.1**. And the graph size dividing the number of edges $\frac{74338.8}{261993} \approx 0.284$ is less than the expected time estimated in **Corollary 5.1**.

Chapter 6

Conclusion and Future Research

6.1 Summary

We introduced the additive group Jacobian on a hyperelliptic curve and Cantor's algorithm for computing group law in Chapter 3. For a hyperelliptic curve of genus g over finite field F_q, the group order of Jacobian is $O(q^g)$. And the group order of a elliptic curve over finite field F_q is $O(q)$. Therefore, the advantage of hyperelliptic curves over elliptic curves is that a smaller base field can be used in order to obtain the same level of security. But the disadvantage is that there exists an algorithm, the hyperelliptic index calculus algorithm, solving HCDLP in subexponential time complexity when the genus becomes large enough. Hence, the small genus hyperelliptic curves are preferred for constructing a hyperelliptic curve cryptosystem. According to **Table 4.1**, we can extend **Table 1.1** to the following **Table 6.1**.

In Chapter 4, we described several variations of hyperelliptic index calculus algorithm. The settings of test data are given in section 4.3. And a computational comparison between these variations is shown in **Table 4.2**.

We also proposed a better algorithm for solving genus 2 HCDLP in Chapter 5. The implementation results can be found in section 5.4. In **Table 5.1**, detailed comparisons between our algorithm and Pollard's rho algorithm are given. It is shown that our algorithm is faster than Pollard's rho algorithm in practice.

Table 6.1 Suggested key size for hyperelliptic curve cryptography.

Security (bits)	Minimum size (bits) of public keys				
	ECC	HECC			
		Genus 2	Genus 3	Genus 4	Genus 5
80	160	80	60	54	50
112	224	112	84	75	70
128	256	128	96	86	80
192	382	192	144	128	120
256	512	256	192	171	160

6.2 Future work

There are several interesting topics for further research.

1. **Solving large sparse linear system over finite field:**

   This is one of the crucial parts in the index calculus algorithm. An improvement of the algorithm for solving large sparse linear system over finite field implies an improvement of the index calculus algorithm.

2. **Reduce the space requirement**

   The disadvantage of our algorithm compared with Pollard's rho method is the space requirement. It takes O(q) memory space in our algorithm. Perhaps, there are other methods which can save the space requirement.

3. **Algorithm design:**

   Design a systematic index calculus algorithm which can extensively use more large primes without much overhead. And analyze how many large primes is the optimal value for collecting enough smooth relations.

Bibliography

[1] L. Adleman, J. DeMarrais and M. Huang, "A Subexponential Algorithm for Discrete Logarithms over the Rational Subgroup of the Jacobians of Large Genus Hyperelliptic Curves over Finite Fields," *Algorithmic Number Theory*, LNCS 877 (1994), 28-40.

[2] D. Cantor, "Computing in the Jacobian of a Hyperelliptic Curve," *Mathematics of Computation*, 48 (1987), 95-101.

[3] David G. Cantor and Hans Zassenhaus, "A New Algorithm for Factoring Polynomials Over Finite Fields," *Mathematics of Computation*, 36:587-592, 1981.

[4] H. Cohen and G. Frey, *Handbook of Elliptic and Hyperelliptic Curve Cryptography*, Chapman & Hall/CRC, 2006.

[5] D. Coppersmith, "Solving Linear Equations over GF(2) via Block Wiedemann Algorithm," *Math. Comp.*, 62(205):333-350, 1994.

[6] A. Enge, "Computing Discrete Logarithms in High-genus Hyperelliptic Jacobians in Provably Subexponential Time," *Math. Comp.*, 71, no. 238, pp. 729-742, 2002.

[7] A. Enge and P. Gaudry, "A General Framework for Subexponential Discrete Logarithm Algorithms", *Acta Arithmetica*, 102 (2002), 83-103.

[8] P. Flajolet, D. Knuth and B. Pittel, "The First Cycles in an Evolving Graph," *Discrete Math.*, 75:167-215, 1989.

[9] R. Flassenberg and S. Paulu, "Sieving in function fields," *Experimental Mathematics*, 8, No. 4, 339-349, 1999.

[10] John B. Fraleigh, *A First Course in Abstract Algebra*, seventh edition, Addison-Wesley, 2003.

[11] W. Fulton, *Algebraic Curves*, Benjamin, New York, 1969.

[12] S.D. Galbraith and N.P. Smart, "A Cryptographic Application of Weil Descent,"

Cryptography and Coding, 7th IMA Conference. LNCS 1746, pp. 191–200. Springer-Verlag, Berlin, 1999.

[13] P. Gaudry, "An Algorithm for Solving the Discrete Log Problem on Hyperelliptic Curves," *Advances in Cryptology —EUROCRYPT 2000*, LNCS 1807 (2000), 19-34.

[14] P. Gaudry and R. Harley, "Counting Points on Hyperelliptic Curves over Finite Fields," *Algorithmic Number Theory —ANSI-IV*, LNCS 1838 (2000), 313-332.

[15] P. Gaudry, F. Hess, and N. Smart, "Constructive and Destructive Facets of Weil Descent on Elliptic Curves," *Journal of Cryptology*, 15:19-46, 2002.

[16] P. Gaudry and E. Thomé, "A Double Large Prime Variation for Small Genus Hyperelliptic Index Calculus," *Crypto ePrint Archive, Report 2004/153*.

[17] C. Guyot, K. Kaveh, V.M. Patankar, "Explicit Algorithm for The Arithmetic on The Hyperelliptic Jacobians of Genus 3," *Journal of Ramanujan Mathematical Society*, 19 (2004), No.2, 119-159.

[18] M. Jacobson and A. van der Poorten, "Computational Aspects of NUCOMP," *Algorithmic Number Theory —ANTS-IV*, LNCS 2369 (2002), 120-133.

[19] N. Koblitz, "Elliptic Curve Cryptosystems," *Mathematics of Computation*, 48 (1987), 203-209.

[20] N. Koblitz, "Hyperelliptic Cryptosystems," *Journal of Cryptology*, 1 (1989), 139-150.

[21] B. A. LaMacchia and A. M. Odlyzko, "Solving Large Sparse Linear Systems over Finite Fields," In A. J. Menezes and S. A. Vanstone, editors, *Advances in Cryptology*, volume 537 of *Lecture Notes in Comput. Sci.*, pages 109–133. Springer–Verlag, 1990. Proc. Crypto '90, Santa Barbara, August 11–15, 1988.

[22] T. Lange, "Efficient Arithmetic on Genus 2 Hyperelliptic Curves over Finite Fields via Explicit Formulae," *Cryptology ePrint Archive: Reprot 2002/121*, 2002.

[23] Niels Lubbes, “The Hyperelliptic Curve Discrete Logarithm Problem,” Master’s thesis, Universiteit van Amsterdam, 2004.

[24] A. Menezes, *Elliptic Curve Public Key Cryptosystems*, Kluwer Academic Publishers, 1993.

[25] A. Menezes, Y. Wu and R. Zuccherato, “An Elementary Introduction to Hyperelliptic Curves” appendix in *Algebraic Aspects of Cryptography* by N. Koblitz, Springer-Verlag, 1998, 155-178.

[26] V. Muller, A. Stein, and C. Thiel, “Computing Discrete Logarithms in Real Quadratic Congruence Function Fields of large genus,” *Math. Comp.*, 68(226):807–822, 1999.

[27] D. Mumford, *Tata Lectures on Theta* II, Birkhauser, Boston, 1984.

[28] K. Nagao, “Improvement of Thériault Algorithm of Index Calculus for Jacobian of Hyperelliptic Curves of Small Genus,” *Cryptology ePrint Achieve*, Report 2004/161.

[29] J. Pelzl, T. Wollinger, and C. Paar, “Low cost security: Explicit formulae for genus-4 hyperelliptic curves,” In M. Matsui and R. Zuccherato, editors, Selected Areas in Cryptography -- SAC 2003, volume 3006 of LNCS, pages 1--16. Springer-Verlag, 2004.

[30] Sakai, Y., and K. Sakurai, “On the Practical Performance of Hyperelliptic Curve Cryptosystems in Software Implementation,” *IECE Trans. Fundamentals*, vol. E83-A, No. 4, April 2000.

[31] Victor Shoup, NTL: A Library for doing Number Theory, available on web http://shoup.net/ntl/.

[32] N. Thériault, “Index Calculus Attack for Hyperelliptic Curves of Small Genus,” *Advances in Cryptology —ASIACRYPT 2003*, LNCS 2894 (2003), 75-92.

[33] D. H. Wiedemann, “Solving Sparse Linear Equations over Finite Fields,” *IEEE*

Trans. Inform. Theory, IT-32(1):54-62, 1986.